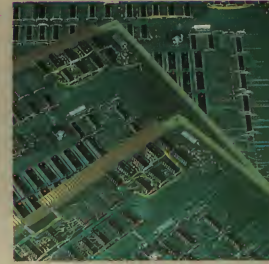


Prime Computer, Inc.

# PRIME

**DOC4130-190P**  
**Prime User's Guide**  
**Revision 19.0**







# Prime User's Guide

**DOC4130-190**

**Third Edition**

**by**

**Anne W. Patenaude**

**This guide documents the software operation of the Prime Computer and its supporting systems and utilities as implemented at Master Disk Revision Level 19 (Rev. 19).**

**Prime Computer, Inc.  
500 Old Connecticut Path  
Framingham, Massachusetts 01701**



## COPYRIGHT INFORMATION

The information in this document is subject to change without notice and should not be construed as a commitment by Prime Computer Corporation. Prime Computer Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may be used or copied only in accordance with the terms of such license.

Copyright © 1982 by  
Prime Computer, Incorporated  
500 Old Connecticut Path  
Framingham, Massachusetts 01701

PRIME and PRIMOS are registered trademarks of Prime Computer, Inc.

PRIMENET, RINGNET, PRIME INFORMATION, and THE PROGRAMMER'S COMPANION are trademarks of Prime Computer, Inc.

TELENET is a registered trademark of Telenet Communications Corporation.

## HOW TO ORDER TECHNICAL DOCUMENTS

### U.S. Customers

Software Distribution  
Prime Computer, Inc.  
1 New York Ave.  
Framingham, MA 01701  
(617) 879-2960 X2053, 2054

### Customers Outside U.S.

Contact your local Prime  
subsidiary or distributor.

### Prime Employees

Communications Services  
MS 15-13, Prime Park  
Natick, MA 01760  
(617) 655-8000, X4837

### PRIME INFORMATION

Contact your PRIME  
INFORMATION dealer.



## PRINTING HISTORY — PRIME USER'S GUIDE

<u>Edition</u>	<u>Date</u>	<u>Number</u>	<u>Documents Rev.</u>
*First Edition	January, 1980	IDR4130	17.2
Second Edition	December, 1980	PDR4130	18.1
Third Edition	August, 1982	DOC4130-190	19.0

\*This edition is out of print.

This edition has been completely revised. The revisions reflect both new and changed technical features. Other changes to the book clarify or amplify information from previous editions. Because this entire edition has been so heavily revised, it does not contain change bars in the margin.

## SUGGESTION BOX

All correspondence on suggested changes to this document should be directed to:

Anne W. Patenaude  
Technical Publications Department  
Prime Computer, Inc.  
500 Old Connecticut Path  
Framingham, Massachusetts 01701

# STANDARD REPORT - FORM NO. 1 (Rev. 1-1-60)

Location	Area	Date	Observer
1000	1000	10-10-60	1000
1000	1000	10-10-60	1000
1000	1000	10-10-60	1000

This section is one of four.

This section has been divided into four parts. The first part is a general description of the area. The second part is a description of the vegetation. The third part is a description of the soil. The fourth part is a description of the water.

## STANDARD REPORT

All measurements are suggested except for the one indicated.

1000 1000 1000 1000  
1000 1000 1000 1000  
1000 1000 1000 1000  
1000 1000 1000 1000



# Contents

ABOUT THIS BOOK	xi
PART I - USING PRIME DOCUMENTATION	
1 INTRODUCTION	
What This Book Contains	1-1
How to Use This Book	1-4
How to Use the Rest of Prime's Documentation	1-5
Programmer's Companions	1-12
PART II - WRITING AND RUNNING PROGRAMS	
2 BEFORE YOU GET STARTED	
Introduction	2-1
Introducing PRIMOS	2-2
Using the File System	2-3
System Prompts	2-14
The Notion of Default	2-14
Terminal Keyboard	2-15
Special Terminal Keys	2-16
Special Characters	2-17
Setting Terminal Characteristics	2-19
3 GETTING STARTED WITH PRIMOS	
Introduction	3-1
Accessing the System	3-2
Creating New Files and Directories	3-8
Examining the Contents of Files and Directories	3-9
Changing the Names of File System Objects	3-11
Copying File System Objects	3-11
Deleting File System Objects	3-14
Protecting File System Objects From Accidental Deletion	3-15
Controlling Access to Files and Directories	3-17
Requesting Information with the HELP Command	3-17
Recording Terminal Sessions	3-17



System Information	3-19
Stopping Unwanted Commands	3-19
Completing a Work Session	3-20

#### 4 CREATING AND LISTING FILES

Entering and Modifying Programs	
— the Editor	4-1
Editor's Error Messages	4-5
Basic Editor Commands	4-6
Sample Editing Sessions	4-20
Listing Programs	4-23
Printing Hard Copy With the SPOOL Command	4-24

#### 5 COMPILING PROGRAMS

Introduction	5-1
Invoking the Compiler	5-3
Object Files	5-4
Listing Files	5-5
Cross Reference	5-5
Code Generation	5-5
Loading	5-6
Compiler Messages	5-7
Combining Languages in a Program	5-7

#### 6 LOADING PROGRAMS

Introduction	6-1
SEG	6-1
Using SEG Under PRIMOS	6-2
Normal Loading	6-3
The R-Mode Loader	6-6
Using the Loader Under PRIMOS	6-7
Normal Loading	6-7

#### 7 RUNNING PROGRAMS INTERACTIVELY

Introduction	7-1
Program Environments	7-1
Executing Programs	7-2
Executing Segmented Runfiles	7-3
Executing R-Mode Runfiles	7-4
Run-time Error Messages	7-5



## 8 DEBUGGING PROGRAMS

Introduction	8-1
Using DBG	8-4
Starting Program Execution	8-4
Stopping Execution	8-5
Examining and Modifying Data	8-7
Examining the Source Code	8-8
Sample Program Debugging Session	8-11

## 9 THE BASICS OF CPL

What is CPL?	9-1
Learning CPL	9-1
How Does CPL Work?	9-2
Creating and Executing CPL Programs	9-2
Debugging CPL Programs	9-4
Using PRIMOS Commands in CPL Programs	9-5
CPL Directives	9-7
Using Variables in CPL Programs	9-9
Decision-making (Branching) in CPL Programs	9-12
&Do Groups	9-15
Using Functions in CPL Programs	9-16
Using CPL with Subsystems: &Data Groups	9-17
How CPL Programs End: The &Return Directive	9-20

## 10 COMMAND FILES AND PHANTOMS

Introduction	10-1
Command File Requirements	10-1
The COMINPUT Command	10-2
The COMOUTPUT Command	10-6
Using DATE and TIME in Command Files	10-8
Phantom Users	10-10

## 11 BATCH JOB PROCESSING

Introduction	11-1
Using the Batch Subsystem	11-2
Submitting Batch Jobs	11-2
Supplying Options via the \$\$ Command	11-5
Controlling Batch Jobs	11-6
Monitoring Batch	11-8



## PART III - SYSTEM FACILITIES

### 12 FILE-HANDLING UTILITIES

Introduction	12-1
Sorting Files (SORT)	12-1
Comparing Files (CMPF)	12-7
Merging Text Files (MRGF)	12-9
Joining Several Files Sequentially (CONCAT)	12-12

### 13 USING TAPES AND CARDS

Accessing Data on Tapes and Cards	13-1
Reading Punched Cards	13-2
Reading Punched Paper Tape	13-4
Magnetic Tape Operations	13-4
Using ASSIGN	13-9
Releasing a Tape Drive (UNASSIGN)	13-12
Magnetic Tape Utilities	13-14
The MAGNET Utility	13-14
Saving Disk Files on Tape (MAGSAV)	13-15
Restoring Files From Tape to Disk (MAGRST)	13-19

### 14 USING PRIMENET

Introduction	14-1
Remote Login	14-2
Attaching to Remote Directories	14-3
Accessing Remote Systems and Networks With NETLINK	14-5
Transferring Files Between Systems (FTR)	14-9
Using Remote Ids	14-19

### 15 SUBROUTINE LIBRARIES

Introduction	15-1
Applications Libraries	15-3
Sort and Search Libraries	15-10
System Libraries	15-13
Condition Mechanism Subroutines	15-18

## PART IV - THE COMMAND ENVIRONMENT

### 16 PROTECTING YOUR FILES AND DIRECTORIES

Introduction	16-1
What Are Access Control Lists (ACLs)	16-2
Specific ACLs and Access Categories	16-5



Default Protection	16-9
Who May Distribute Rights	16-14
Commands for Using ACLs and Access Categories	16-16
Listing Access Rights	16-16
Controlling Access to Files and Directories	16-18
Changing Access Rights	16-22
Priority ACLs	16-24
Tips on Setting Access Rights	16-25
ACL Subroutines	16-27
 17 CUSTOMIZING YOUR ENVIRONMENT	
Introduction	17-1
Changing the Prompt Message (RDY)	17-2
Creating and Using Abbreviations (ABBREV)	17-3
Using Global Variables	17-9
Creating Login Files	17-10
Sending Messages	17-14
Using Disk Quotas	17-18
 18 COMMAND LINE FEATURES	
Introduction	18-1
Iteration	18-1
Wildcarding	18-4
Treewalking	18-11
Name Generation	18-19
Combining Command Line Features	18-22
 19 COMMAND LINE PROCESSING	
Introduction	19-1
Abbreviation Expansion	19-2
Syntax Suppression	19-3
Multiple Command Processing	19-4
Variable and Function Evaluation	19-4
Iteration	19-5
Treewalking	19-5
Wildcarding	19-6
Name Generation	19-7
Execution	19-7
An Example	19-7
 20 USING THE CONDITION MECHANISM	
Introduction	20-1
Using the Condition Mechanism	20-2
The System Default On-unit	20-2
On-unit Actions	20-3
Writing On-units	20-4



Scope of ON-units	20-5
A FORTRAN Example	20-6

## APPENDIXES

A GLOSSARY OF PRIME CONCEPTS AND CONVENTIONS	A-1
B SYSTEM DEFAULTS AND CONSTANTS	B-1
C ASCII CHARACTER SET	
Prime Usage	C-1
ASCII Character Set (Non-Printing)	C-2
ASCII Character Set (Printing)	C-4
D ERROR MESSAGES	
Introduction	D-1
SEG Loader Error Messages	D-2
LOAD Loader Error Messages	D-7
Run-time Error Messages	D-10
Batch Warnings and Messages	D-32
Conditions Flagged by the Condition Mechanism	D-42
E EDITOR COMMAND SUMMARY	E-1
F THE PASSWORD PROTECTION SYSTEM	
Introduction	F-1
Assigning Directory Passwords	F-2
Using Passwords to Gain Access to Directories	F-3
Setting Access Rights on File System Objects	F-3
Converting a Password Directory to an ACL Directory	F-4
Converting an ACL Directory to a Password Directory	F-6
Creating a Password Subdirectory under an ACL Directory	F-7
G SYSTEM INFORMATION	G-1
INDEX	X-1



# About This Book

The Prime User's Guide introduces the new user to PRIMOS (Prime's operating system), and to Prime's file system, utilities, compilers, and subroutine libraries. The presentation is largely tutorial and centers on that small portion of the software that does most of the application programmer's work. References to other Prime documents tell you where to find detailed information.

This book assumes that you have some programming background in a high-level language. It does not assume any familiarity with Prime equipment or software.

## OTHER USEFUL BOOKS FOR NEW USERS

Other useful books for new users of Prime systems include:

- New User's Guide to Editor and Runoff, which details how to use Prime's line-oriented text editor (ED) and text formatter (RUNOFF).
- PRIMOS Command Reference Guide, which contains information on format and usage of all PRIMOS user commands.
- CPL User's Guide, which explains the full usage of Prime's command procedure language.
- The reference guide(s) for the programming language(s) you will use on your Prime system.



## PRIME DOCUMENTATION CONVENTIONS

The following conventions are used in command formats, statement formats, and in examples throughout this document.

<u>Convention</u>	<u>Explanation</u>	<u>Example</u>
UPPERCASE	In command formats, words in uppercase indicate the actual names of commands, statements, and keywords. They can be entered in either uppercase or lowercase.	SLIST
abbreviations	If a command or statement format has an abbreviation, it is indicated by an <u>underlining</u> . However, where this cannot be done clearly, the abbreviation either appears below the command with both enclosed in braces, or on a line beneath the format and introduced by the word: "Abbreviation".	<u>LOGOUT</u>  {CHANGE_PASSWORD} {CPW}  SET_ACCESS Abbreviation: SAC
lowercase	In command formats, words in lowercase indicate variables, items for which the user must substitute a suitable value.	LOGIN user-id
<u>underlining</u> in examples	In examples, user input is underlined, but system prompts and output are not.	OK, <u>DATE -MONTH</u> September OK,
Brackets [ ]	Brackets enclose a list of one or more optional items. Choose none, one, or more of these items (0-n).	SPOOL[-LIST -CANCEL]



Default  
Indicator  
•

In a list of options, the default choice, if one exists, is indicated by a bullet "•". If the user selects no options, the system automatically chooses the default option.

STATUS [ ALL •  
DISKS  
NETWORKS  
UNITS  
USERS ]

Braces  
{ }

Braces enclose a vertical list of items. Choose one and only one of these items.

CLOSE { filename }  
-ALL }

Ellipsis  
...

An ellipsis indicates that the preceding item may be repeated.

item-x[,item-y]...

Parentheses  
( )

In command or statement formats, parentheses must be entered exactly as shown.

DIM array (row,col)

Hyphen  
-

Wherever a hyphen appears in a command line option, it is a required part of that option.

SPOOL -LIST

(CR)

The (CR) symbol indicates a single carriage return which is generated on most terminals by hitting the RETURN key.

#### ADDITIONAL PRIME USER'S GUIDE CONVENTIONS

'

An apostrophe before a number indicates that octal notation follows.

'210

Oval

Ovals are used in diagrams to represent files.



Rectangle

Rectangles are used in diagrams to represent directories.





Hexagon

Hexagons are used in diagrams to represent segment directories.



Triangle

Triangles are used in diagrams to represent access categories.



### COMMAND LINE FORMAT

The PRIMOS command line has the following basic format:

command [argument]... [option [argument]]...

A command is a PRIMOS-recognized word that signals a certain action. An option is a PRIMOS-recognized word preceded by a hyphen and related to a command. It further defines the action the command should take. An argument is a variable and may be used with either a command or an option. An argument may set a value for an option, or it may specify an object on which the command or option acts.

One or more spaces must to separate each command line element (command, argument, or option) from the next.

The following example illustrates a command line:

LOGIN TERRY -ON SYSX

In this example:

LOGIN is the command

TERRY is an argument for the command LOGIN

-ON is an option for the command LOGIN

SYSX is an argument for the option -ON



**PART I**

**Using Prime Documentation**

(PART I)

1. The first of the following is a list of the names of the persons who have been elected to the office of Mayor of the City of New York since the year 1898.



# 1

## Introduction

### WHAT THIS BOOK CONTAINS

The Prime User's Guide is an introduction and overview to programming in a high-level language on a Prime computer. It contains all the information new users need to get started on a Prime system. The book also tells new and experienced users alike what is available on Prime computers and where to locate information.

### Changes in This Edition

Chapters 3, 8, 14, 16, 17, 18, and 19 are new or totally revised. Appendixes F and G are new. The remainder of the book is also heavily revised. Revisions reflect both technical changes and rewrites to clarify or amplify existing information. Because of the magnitude of these changes, this edition does not contain changes bars in the margin.

### Organization of This Book

This guide is divided into four parts.

Part I contains an introduction (Chapter 1), which tells how to use this book and provides an annotated guide to Prime's features and documentation.



Part II introduces users to PRIMOS (Prime's Operating System) and carries them step by step through the acts of creating and running a program, as follows:

- Chapter 2 introduces Prime's operating system, PRIMOS, and its file management system.
- Chapter 3 tells how to access the system: how to log in; how to create, list, copy, and delete files and directories; and how to log out when you are done.
- Chapter 4 explains how to enter files (programs, text files, and data files), using Prime's ED editor; and how to get files printed on the line printer.
- Chapter 5 provides an introduction to compiling programs. Simple programs can be compiled from the information given in this guide. For more complex programs, or programs for which the programmer wishes to use the advanced features of Prime's compilers, the programmer should consult the specific language reference guide.
- Chapter 6 provides an introduction to linking and loading programs with Prime's two loaders, SEG and LOAD. The information in this section enables users to load simple programs. The language guides provide information on language-specific features; the LOAD and SEG Reference Guide provides full information on advanced techniques.
- Chapter 7 provides an introduction to executing programs interactively. (Language-specific details on execution and debugging are in the language guides.)
- Chapter 8 provides an introduction to debugging programs using Prime's Source-Level Debugger (DBG). Full information is provided in the Source Level Debugger Guide.
- Chapter 9 introduces Prime's command procedure language, CPL, and shows how to write command procedure files for interactive or non-interactive running of programs.
- Chapter 10 tells how to create command input and output files for the non-interactive running of programs, how to execute command files from the terminal, and how to execute command files as phantoms (that is, as independent processes not connected with your terminal).
- Chapter 11 provides full information on how to execute programs using Prime's batch processing environment.



Part III provides an introduction to the system utilities available on your Prime system.

- Chapter 12 tells how to use four file-handling utilities:
  - SORT, which sorts and merges files
  - CMPF, which compares files and notes disparities
  - MRGF, which creates one updated file out of several disparate files
  - CONCAT, which joins several files together sequentially
- Chapter 13 explains how to handle magnetic tapes, punched cards, and punched paper tapes on a Prime system.
- Chapter 14 explains PRIMENET, Prime's networking facility, and tells how users can take advantage of it. The chapter also explains how to use the File Transfer Service (FTS) to transfer files between a local and remote site, and how to use remote ids on certain remote systems.
- Chapter 15 provides a selected list of important subroutines and libraries available for use by high-level language programs. Full information on subroutines appears in the Subroutines Reference Guide.

Part IV provides a more advanced look at PRIMOS. In particular, it discusses several ways in which you can alter the command environment on a terminal-by-terminal or program-by-program basis.

- Chapter 16 explains access control lists (ACLs), Prime's mechanism for protecting your work from unauthorized users.
- Chapter 17 shows how to modify the system prompts (RDY); how to define your own abbreviations for PRIMOS commands (ABBREV); how to define global variables to use with PRIMOS and in programs; how to create special command files that execute automatically at login; how to send messages to other users (MESSAGE); how to set limits, or quotas, on the amount of storage space you wish to allot for your subdirectories.
- Chapter 18 explains four enhancements to command line processing:
  - Iteration, for performing the same command with different file system objects.
  - Wildcarding, for instructing PRIMOS (through special symbols) to execute a command on a group of objects without listing each objectname.



- Treewalking, for searching through directory levels and executing the specified command on the appropriate objects.
- Name generation, to avoid repeating long objectnames by telling PRIMOS to substitute the full names for generation symbols.
- Chapter 19 explains the order in which the special features available for the command line are processed and how they interact. The chapter diagrams a long example of a complex command line to illustrate how various command line features behave.
- Chapter 20 explains PRIMOS's condition mechanism and shows how users can write their own on-units (error-handling subroutines).

In addition to the body of the text, this guide provides the following appendixes:

- A glossary of terms used in Prime documentation
- A list of system defaults and constants
- The ASCII character set
- A list of system error messages
- A summary of Editor commands
- A discussion of the password protection system (an alternative to access control lists), and of how to convert protection from passwords to access control lists and vice versa.
- A table of commands that provide useful system information

#### HOW TO USE THIS BOOK

We suggest that you:

- Read Chapters 1 through 4 before beginning to work on the system.
- Read Chapters 5 through 11 before you try to compile, load, or run programs.
- Use Chapters 12 through 15 as reference sections:
  - Chapter 12 if you need to sort, compare, merge, or concatenate files



- Chapter 13 if you need to use magnetic tapes, cards, or paper tape
- Chapter 14 if the computer you work on is part of a network
- Chapter 15 to find out whether PRIMOS has a subroutine or utility that does some task you need to do, or whether you'll have to write your own
- Read Chapters 16 through 20 when you have become somewhat familiar with the system, to discover some more sophisticated conveniences PRIMOS can offer you.
- Refer to the glossary in Appendix A if you encounter terms you do not recognize.

#### HOW TO USE THE REST OF PRIME'S DOCUMENTATION

The rest of this chapter supplies a brief guide to other Prime documentation. These books give detailed information about software that is not discussed in the Prime User's Guide or which is introduced in a general way only. (Titles followed by asterisks document separately priced products.)

#### The Central Guides

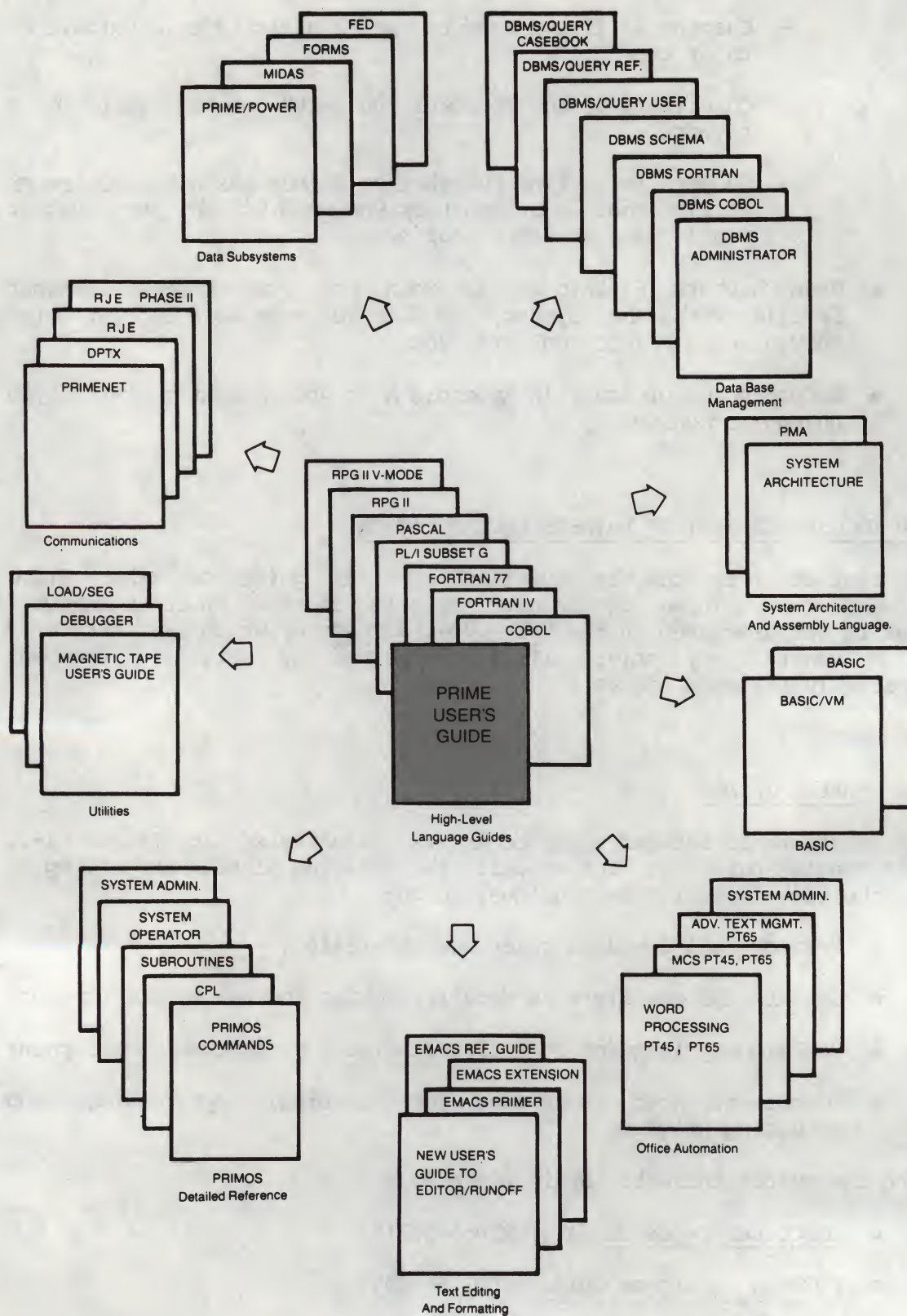
The relationship between these books is illustrated in Figure 1-1. This user's guide is the center: the starting place. Supporting it are the high-level language guides, which:

- Provide full language reference materials
- Explain the compilers in detail, showing the use of all options
- Explain any language-specific techniques of program development
- Discuss advanced techniques for loading, optimizing, and debugging programs

Language guides currently available are:

- COBOL Reference Guide (FDR3056-101)\*
- FORTTRAN Reference Guide (FDR3057-101)
- FORTTRAN 77 Reference Guide (DOC4029-183)\*
- Pascal Reference Guide (IDR4303)\*





Organization of Prime Software Documentation  
Figure 1-1



- PL/I Subset G Reference Guide (IDR4031)\*
- RPG II Reference Guide (PDR3031) and the RPG II Debugging Template (FDR3275)
- RPG II V-Mode Compiler Reference Guide (IDR5040)\*

### PRIMOS Detailed References

The commands explained in this guide and the language guides will carry most applications programmers through most of their work. For those who need more detailed references about PRIMOS, each topic discussed in this book is treated more fully in our reference guides. The reference guides that applications programmers are most likely to use are:

- PRIMOS Commands Reference Guide (FDR3108-190)

This guide discusses all PRIMOS level commands available to the user.

- CPL User's Guide (DOC4302-190)

This guide describes how to use Prime's Command Procedure Language.

- Subroutines Reference Guide (DOC3621-190)

This guide tells how to incorporate into your own programs the various subroutines supplied by Prime.

A reference guide for system administrators is:

- System Administrator's Guide (DOC5037-190)

This guide tells how to configure, bring up, and administer a Prime system.

A reference guide for system operators is:

- System Operator's Guide (DOC5038-190)

This guide tells how to maintain a Prime system and respond to users' needs.



## Utilities

Utilities explained in this guide are discussed in greater detail in several books, which may be of interest to programmers:

- LOAD and SEG Reference Guide (PDR3524)

This guide provides a full discussion of Prime's loaders for users interested in taking advantage of their advanced features.

- Source Level Debugger Reference Guide (IDR4033)\*

This guide provides both introductory and full discussions on the use of Prime's interactive debugger for FORTRAN, FORTRAN 77, and PL/I Subset G programs.

- Magnetic Tape User's Guide (DOC5027-183)

This guide provides information on the use of magnetic tapes for backing up, archiving, transferral of information between machines, and translations to and from non-Prime tape formats.

## BASIC

BASIC is implemented on Prime computers as a fully interactive, self-contained environment. Working in BASIC, a programmer can write, compile, execute, and debug a program while remaining inside the BASIC environment. Prime's guides to working with BASIC, therefore, are similarly self-contained, providing both full explanations of all BASIC features and all introductory material needed to get the new user onto the system. The guides are:

- Interpretive BASIC Programmer's Guide (IDR1813)
- BASIC/VM Programmer's Guide (FDR3058-101)\*

## Assembly Language

For assembly language programmers, and for those interested in learning about Prime's computer architecture, there are:

- Assembly Language Programmer's Guide (FDR3059-101)
- System Architecture Reference Guide (PDR3060-182)



Text Editing

For users concerned with text editing, Prime has two editors: the ED line-oriented editor and the EMACS screen-oriented editor. The product RUNOFF allows you to create formatted printouts. The following books discuss these products:

- New User's Guide to Editor and Runoff (FDR3104-101)

This guide explains in full detail Prime's editor (ED) and its text formatting utility (RUNOFF). (Aimed at users who may not be programmers, this guide also provides a less technical introduction to Prime software for secretaries, typists and data entry personnel.)

- EMACS Primer (IDR6107)\*

The EMACS Primer presents a basic subset of commands for the EMACS screen editor. It is designed as a tutorial for the non-technical user.

- EMACS Reference Guide (IDR5026)\*

This guide describes the full use of the EMACS screen editor. It is intended for users at any level, but assumes familiarity with the EMACS Primer or the EMACS PT45 Primer.

- EMACS Extension Writing Guide (IDR5025)\*

This book is aimed at a sophisticated EMACS user who is also a programmer. It shows programmers how to create new editing commands and alter existing commands to fit their particular needs.

Data Subsystems

- PRIME/POWER Guide (PDR3709)\*

POWER is an easy-to-use data management system with English-like commands that allow the user to create, access, update, and report on MIDAS, ASCII, or binary files. POWER files are compatible with (and can be accessed from) BASIC/VM, COBOL, and FORTRAN programs.

- MIDAS User's Guide (IDR4558-176)

MIDAS (Multiple Index Data Access System) creates and maintains keyed-index data files to hold large amounts of information in a quickly accessible format. MIDAS files are handled through a variety of high-level language interfaces and are useful to applications programmers.



- FORMS Programmer's Guide (PDR3040)\*

FORMS allows applications programmers to design screen formats (such as representations of business forms), to store the formats in a directory and to write applications programs that use these screen formats to facilitate data entry.

- FED User's Guide (IDR4940)\*

FED (Forms Editor) allows non-programmers to design screen formats (such as representations of business forms). FED allows non-technical users to create forms on the terminal screen, generate a FORMS definition language source file automatically, and compile the form so that it can be used by Prime's programming languages.

### Data Base Management

Seven guides document Prime's data base management system.

Any data base user, without knowledge of programming, may use Prime's DBMS/QUERY books to retrieve information from a data base constructed with Prime's DBMS and format this information into reports. The books are:

- DBMS/QUERY User's Guide (IDR4608)\*
- DBMS/QUERY Reference Guide (IDR4607)\*
- DBMS/QUERY Report Generator Casebook (IDR5650)\*

Programmers writing data base applications programs in FORTRAN or COBOL should consult:

- DBMS FORTRAN Reference Guide (PDR3045)\*
- DBMS COBOL Reference Guide (PDR3046)\*

Data base administrators concerned with setting up and maintaining a data base should refer to:

- DBMS Administrator's Guide (PDR3276)\*
- DBMS Schema Reference Guide (PDR3044)\*

### Communications

If you are installing or using a network, or if you are writing programs concerned with network functions, the guide you want is:

- PRIMENET Guide (DOC3710-190)\*



If your installation has (or will have) DPTX (Distributed Processing Terminal Executive), and you are involved with it, you will want:

- Distributed Processing Terminal Executive Guide (IDR4035)\*

If your work involves any of the remote batch terminal emulators - HASP, RJE2780, RJE3780, 200UT, 1004, GRIS, or ICL 7020 - you can find out how to handle them in:

- Remote Job Entry Guide (IDR4036)\*
- Remote Job Entry Phase II Guide (DOC6053-190)\*

### Office Automation

Prime's Office Automation System is currently supported by the following documents:

- OAS Word Processing Guide (PT45) (IDR5020)\*
- OAS Word Processing Guide (PT65) (IDR5021)\*

These books provide complete instructions for the Word Processing module of Prime's Office Automation System for either PT45 or PT65 terminals.

- OAS Management Communications and Support Guide (PT45) (IDR5022)\*
- OAS Management Communications and Support Guide (PT65) (IDR5023)\*

These books provide instructions for the Management Communications and Support module of Prime's Office Automation System on either PT45 or PT65 terminals. This module includes electronic mail, correspondence management, and management support functions.

- OAS Advanced Text Management Guide, (PT65) (IDR4353)\*

This book provides complete instructions for the Advanced Text Management module of Prime's Office Automation System. Advanced Text Management enhances Word Processing by providing automated proofreading and hyphenation, and word-for-word translation in up to four languages is included.

- OAS System Administrator's Guide (IDR4354)\*

This book provides instructions on management of Prime's Office Automation System. Such items as creation of user ids, generation and purging of schedule grids, and printing hard copies of system reports are included.



PROGRAMMER'S COMPANIONS

Prime also provides a series of handy pocket-sized reference summaries on many of its products. The following titles are currently available:

- PRIMOS Commands Programmer's Companion (FDR3250)
- FORTRAN Programmer's Companion (FDR3338)
- FORTRAN 77 Programmer's Companion (FDR4030)\*
- COBOL Programmer's Companion (FDR3339)\*
- BASIC/VM Programmer's Companion (FDR3341)\*
- Assembly Language Programmer's Companion (FDR3340)
- Loading and Debugging Programmer's Companion (FDR5094)
- System Administrator Programmer's Companion (FDR3622)
- PRIME/POWER Companion (FDR4034)\*



**PART II**

**Writing and Running Programs**







# 2

## Before You Get Started

### INTRODUCTION

Before you begin using your Prime computer, you'll need to know:

- A few facts about Prime's operating system, PRIMOS
- A few facts about the PRIMOS file management system
- What the system prompts are
- How to use the terminal
- What meaning the special terminal keys have for the PRIMOS operating system and for some of its subsystems
- What meaning some special characters have for PRIMOS and for some of its subsystems
- How to define your own special characters or change the characteristics of your terminal

This chapter explains all of these features and functions, in the above order. (You may also wish to examine the list of Prime Documentation Conventions at the beginning of this book.)



## INTRODUCING PRIMOS

All Prime 50 series computers use a common operating system known as PRIMOS. PRIMOS keeps track of who is allowed to use the computer, of what work users are doing on the computer, and of where their work is stored. Users type commands to PRIMOS at the terminal, and PRIMOS performs the work.

Depending on which central processing unit you have, a Prime Computer under PRIMOS can support up to 128 simultaneous users. Each user is totally independent. Each one may use any utility (such as an editor or compiler), and may write, compile, load, and execute any program, in any language, without regard to what other users are doing on the system.

## Compatibility

Because a common operating system is used throughout the Prime processor line, programs created on one Prime computer can be used on most other Prime computers, without modification. There is complete upward compatibility among all models. Considerable downward compatibility exists among other models as well, as long as system constraints on program size and mode of code generated are observed.

## Some Hardware Features

Prime's hardware supports this multi-user, interactive environment with:

- Virtual memory, which allows users to run programs larger than the physical memory of the machine. A program may be as large as 32 megabytes on the Prime 450 and up.
- Segmentation of programs, allowing the separation of code and data. This facilitates the creation of pure code for shared or recursive procedures.
- A ring protection system which provides hardware protection for the operating system and user subsystems.

Except for segmentation of large programs, users have little immediate concern with these features. They are largely invisible, designed to let users concentrate on their own goals without worrying about the hardware.



USING THE FILE SYSTEMFile System Objects

Information is stored in Prime computers in file system objects. A file system object is a collection of data that has its own name. It may be referenced by users through this name. File system objects are of four types:

- File
- Directory
- Segment directory
- Access category

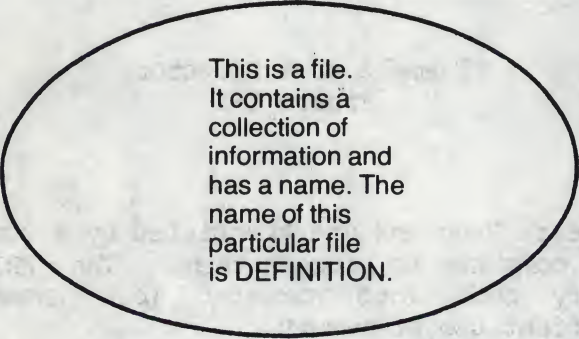
Each of these four is explained in the sections that follow and is illustrated by a diagram.

Note

The term "file system object", or "object", should be distinguished from an "object file" (also called a "binary file"). An object file is created by a compiler when the compiler translates a source program written in a high-level language into binary code (also called object code). Object files are more fully discussed in Chapter 5.

Files: A PRIMOS file is an organized collection of information identified by a filename. The file contents may represent a source program, the text of a document, or anything the user can express in the available keyboard symbols. Figure 2-1 represents a file.

## DEFINITION



This is a file.  
It contains a  
collection of  
information and  
has a name. The  
name of this  
particular file  
is DEFINITION.

Example of a File  
Figure 2-1



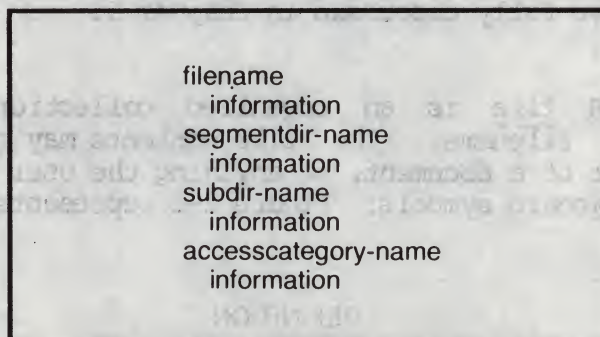
**Directories:** A directory is a list of names of file system objects, along with information on their location and their characteristics (such as their date and time last modified). Each directory has a name. If no objects are listed in the directory, it is called an "empty directory," but it still exists. Figure 2-2 illustrates a directory.

"Directory" is a generic term. There are three kinds of directories:

- Master File Directory (MFD)
- User File Directory (UFD)
- Subdirectory (sub-UFD)

The disk is divided into a group of "top-level" directories; each is called a User File Directory (UFD). The special directory that contains the listing of UFDs is called the Master File Directory (MFD), which is a partition on a disk. UFDs may contain the names of other directories, called subdirectories (sub-UFDs). A sub-UFD may also contain more sub-UFDs. This hierarchical structure of PRIMOS is explained further in the section Tree Structure, and is illustrated in Figure 2-6.

#### DIRECTORY.A



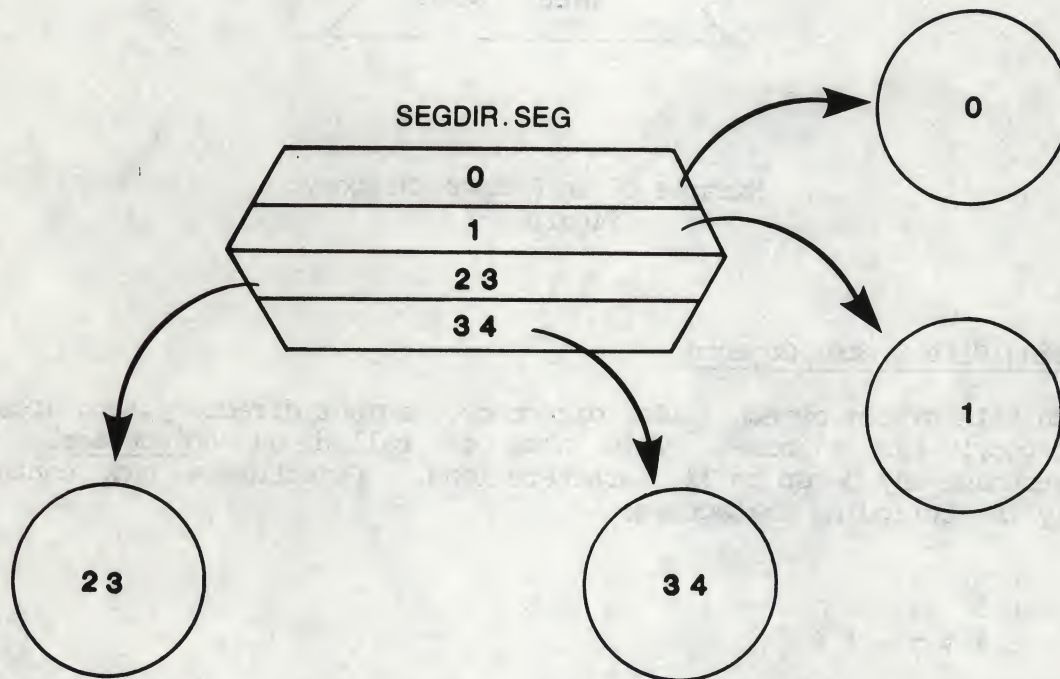
Example of a Directory  
Figure 2-2

**Segment Directories:** Programs are translated by a compiler into binary code to allow the computer to process them. The PRIMOS SEG utility divides the binary code into numbered (but unnamed) sections to maximize the efficient use of memory.



The locations of all these sections, together with other information, are stored in a named special listing, which is called a segment directory. Figure 2-3 illustrates a segment directory. Segment directories are discussed further in the Subroutines Reference Guide.

Segment directories are never called simply "directories" in this book. They are always called "segment directories."



The hexagonal figure represents a segment directory named SEGDIR.SEG. It contains the location of each of the blocks of information represented by circles. These blocks of information do not have names, but do have numbers (not necessarily consecutive).

Example of a Segment Directory  
Figure 2-3

Access Categories: An access category is a special list of user-ids and the particular access rights they have (such as "JOHN" has "ALL" rights). Access categories are part of the PRIMOS protection system to prevent unauthorized users from accessing your work. Figure 2-4 illustrates an access category. (The \$REST designation in the figure means "everybody else".) Access categories are explained in detail in Chapter 16.



PROTECT.ACAT



USER-ID: ALL  
\$REST: NONE

Example of an Access Category  
Figure 2-4

### Naming File System Objects

Each file system object (file, directory, segment directory, and access category) has a name. This name is called an objectname. An objectname may be up to 32 characters long. Objectnames may contain only the following characters:

A-Z  
0-9  
\_ # \$ - . \* & /

The first character of an objectname may not be numeric. On some devices underscore (\_) prints as a backarrow(←).

Because they may cause problems of confusion with certain commands or command syntaxes, names beginning with "\_", "&", "\$", and ".", as well as the name "\*", should be avoided.

Suffix Conventions: Prime's suffix conventions use standard suffixes to objectnames to identify various sorts of file system objects. In particular, suffixes identify various sorts of files.

The naming conventions help you to keep track of the types of files in your directory (for example, a source program or a compiler-generated listing of a source program). They also help you to access groups of files using PRIMOS's wildcard functionality (explained in Chapter 18).

Using these conventions, an objectname is usually divided into two components: the base name and the suffix. A period (.) separates the components.



For example, the following are all objectnames with two components:

TEST.CPL  
PROGRAM.FIN  
SAMPLE.SEG

In these examples, "CPL", "FIN", and "SEG" are all suffixes. There may be up to 16 components in an objectname, separated by periods. However, only the final component is considered to be the suffix. Names with more than three components are not recommended.

Commonly used objectname suffixes that are recognized by Prime software include:

<u>Objectname With Suffix</u>	<u>Meaning</u>
filename.compiler-name	Sourcefile (A full list is in Table 5-1, in Chapter 5.)
filename.LIST	Listing file (created by compiler)
filename.BIN	Binary file (created by compiler)
filename.SAVE	Runfile created by R-mode loader
segdirectoryname.SEG	Segment directory created by SEG
filename.CPL	CPL file
categoryname.ACAT	Access category

Other common user suffixes, which are not recognized by Prime Software but are recommended to order and clarify your work, include:

<u>Objectname With Suffix</u>	<u>Meaning</u>
filename.ABBREV	Abbreviation file
filename.COMI	Command input file (or phantom command file)
filename.COMO	Command output file
filename.FORM	FORMS file
filename.GVAR	Global variable file
filename.MAP	Map file (created by LOAD or SEG)
filename.RUNI	Runoff Source (input) file
filename.RUNO	Runoff output file
filename.T	Temporary file

A complete list of suffixes appears in the glossary under "objectname conventions."



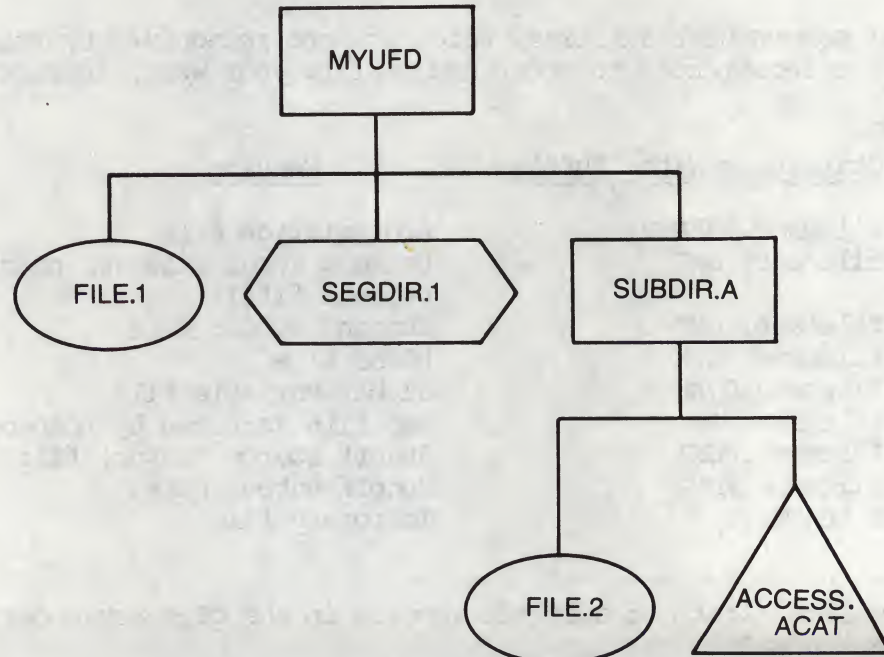
### Storage of File System Objects

Objects are normally stored on the disks attached to the computer system. No detailed knowledge of the physical location of an object is required because the user, through PRIMOS commands, refers to objects by name. On some systems, objects may also be stored on magnetic tape for backup or for archiving.

### Tree Structure

The PRIMOS file directory system is arranged in levels, as a tree. At the base are the disk volumes (also called partitions, logical disks or MFDs). A disk volume may occupy either a complete disk pack or a partition of a multihead disk pack. Each disk volume is an MFD containing the names of several UFDs. Each UFD may contain files, segment directories, access categories, and also other directories called subdirectories or sub-UFDs. These subdirectories may contain any file system object, including more subdirectories. Directories may have subdirectories to any reasonable level.

Figure 2-5 illustrates a simple PRIMOS tree.



A PRIMOS Tree  
Figure 2-5



Pathnames

Pathnames tell PRIMOS which file system object a user wishes to work on in the PRIMOS tree. A pathname (also called a treename) is a name used to specify a particular object. A full pathname consists of the names of the disk volume, the UFD, a chain of subdirectories, and the target object. The disk name is enclosed in angle brackets (<>), and a single right angle bracket (>) separates all other objectnames from one another. For example,

`<FOREST>BEECH>BRANCH5>SQUIRREL`

specifies a file on the disk volume FOREST, under the UFD BEECH and the sub-UFD BRANCH5. The file's name is SQUIRREL.

Figure 2-6 illustrates how pathnames show paths through a tree of directories and files. The shaded areas illustrate the pathname `<FOREST>BEECH>BRANCH5>SQUIRREL`. (Segment directories and access categories do not appear in this diagram, although they could be part of any tree.)

Rules for Specifying Pathnames: No spaces should appear immediately before or after an angle bracket. For example:

Correct: `<HOUSE>DOOR>KNOB`

Incorrect: `<HOUSE >DOOR > KNOB`

Some directories may have a password associated with them (explained in Appendix F). When you use a passworded directory in a pathname, you must give the directory name followed by one blank space and the password:

directoryname password

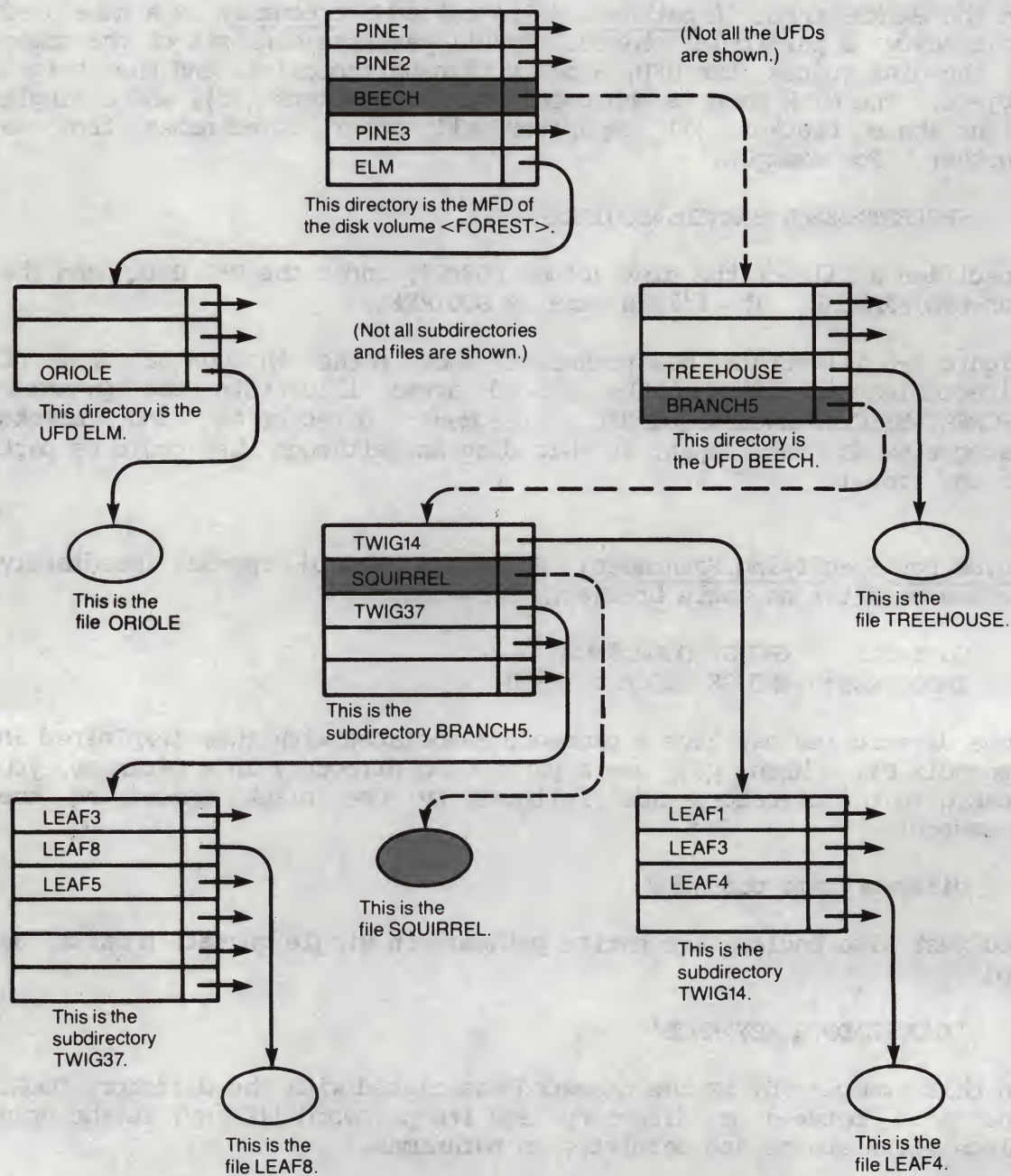
You must also enclose the entire pathname in single quotation marks, as in:

`'<HOUSE>DOOR KEY>KNOB'`

In this example KEY is the password associated with the directory DOOR. The space between a directory and its password (if any) is the only place where spaces are permitted in pathnames.

Uniqueness of Pathnames: Each pathname is unique. This uniqueness means that two objects may not have the same objectname in the same directory; objects in different directories may have the same objectname. Figure 2-7 illustrates legal and illegal possibilities for naming an object "BRANCH2".

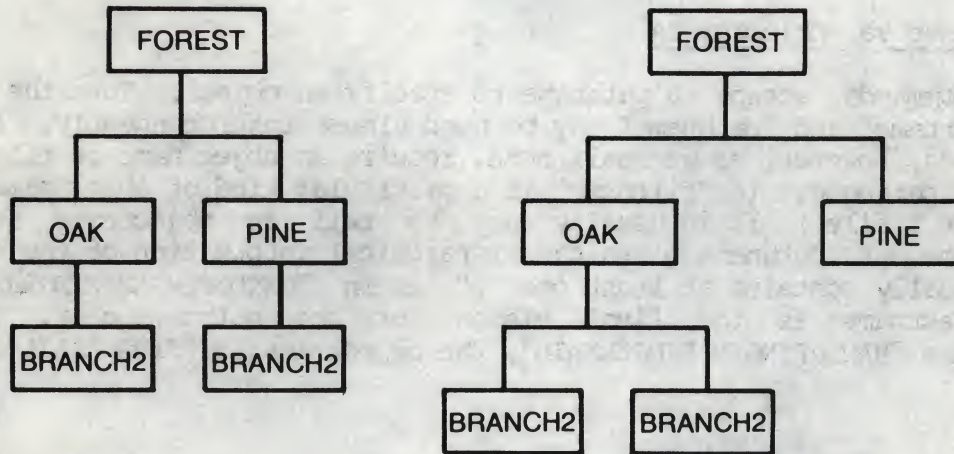




Examples of Files and Directories  
in PRIMOS Tree-structured File System

Figure 2-6





Legal. Two distinct  
pathnames exist:  
<FOREST>OAK>BRANCH2  
<FOREST>PINE>BRANCH2

Illegal. Only one object  
named BRANCH2 may exist in  
OAK for the pathname:  
<FOREST>OAK>BRANCH2

Legal and Illegal Objectnames  
Figure 2-7

Disk volume names, and their associated logical disk numbers, may be found with the STATUS DISKS command, described in Chapter 14. A pathname can be made with the logical disk number, instead of the disk volume name. For example, if FOREST is mounted as logical disk 3,

<3>BEECH>BRANCH5>SQUIRREL

specifies the same file as the previous example.

Usually each UFD name is unique throughout all the logical disks. In our example that would mean that there would be only one UFD named BEECH in all the logical disks, 0 through 61. When that is the case, the volume or logical disk name may be omitted, and PRIMOS will search all the logical disks, starting from 0, until the UFD is found. For example, if there is no UFD named BEECH on disks 0, 1, or 2, then

BEECH>BRANCH5>SQUIRREL

will specify the same file as the previous two examples. This last form of pathname, in which the disk specifier is omitted, is called an ordinary pathname because it is very frequently used.



Pathnames vs. Objectnames

Most commands accept a pathname to specify an object. Thus the terms "objectname" and "pathname" may be used almost interchangeably. A few commands, however, as we shall note, require an objectname or filename, not a pathname. (A "filename" is a particular kind of objectname, the name of a file.) It is usually easy to tell an objectname from a pathname. A pathname gives the hierarchical information of the object and usually contains at least one ">", as in "HAND>FINGER>FINGERNAIL". An objectname is the final element of the pathname only. In the pathname "HAND>FINGER>FINGERNAIL", the objectname is "FINGERNAIL".

Origin Directory and Current Directory

When you log in, PRIMOS connects, or "attaches", you to a particular directory known as your "origin directory". (When you are attached to it, it is also known as your "current" directory.) This directory may be a UFD or a sub-UFD. Usually, it is a directory to which you have fairly full rights, and in which you do a fair amount of your work. However, you may "leave" this directory and "attach" to another directory. To do this, you use the ATTACH command (described in Chapter 3). Whatever directory you attach to becomes your "current" directory. More than one user can be attached to the same directory at the same time.

Relative Pathnames

It is often more convenient to specify a file or directory pathname relative to the current directory, rather than via a UFD. For example, when the current directory is:

BEECH>BRANCH5

the commands

OK, SLIST BEECH>BRANCH5>TWIG9>LEAF3

and

OK, SLIST \*>TWIG9>LEAF3

have the same meaning. The symbol "\*" as the first directory in a pathname designates everything in the pathname down to and including the current directory.



Current Disk

Occasionally it will be necessary to specify a UFD as being on the disk volume you are currently using; that is, where your current directory is. For example, if another disk volume has UFD names identical to those you wish to use on your current disk, you must specify which disk is to be used, each time a pathname is given. The current disk is specified by: "<\*>". For example:

```
<*>BEECH>BRANCH5
```

Note

Do not confuse "<\*>", meaning current disk, with the "\*" alone, which means everything in the pathname down to and including the current directory.

Password-protected Directories

Occasionally, a directory is protected by having a password. In this case, the password becomes part of the directory name or pathname. The password is entered after the name of the directory to which it belongs, separated by one blank space. Apostrophes enclose the entire pathname.

For example, if the directory BEECH had the password SECRET, a pathname using it might be

```
'BEECH SECRET>BRANCH5'
```

Note

The ACL protection system, explained in Chapter 16, will normally be used instead of the directory password system at Rev. 19.0 and will make pathnames using passwords and apostrophes unnecessary. More information on directory passwords appears in Appendix F.

More Information on the PRIMOS File System

For a more technical description of the PRIMOS file system and a description of the ordering of information within file system objects, refer to the Appendix "File Management System Concepts" in the Subroutines Reference Guide.



## SYSTEM PROMPTS

### The "OK," Prompt

The "OK," prompt indicates that the most recent command to PRIMOS has been successfully executed, and that PRIMOS is ready to accept another command from the user.

Type-ahead: PRIMOS supports type-ahead. The user need not wait for the "OK," after one command before beginning to type the next command. However, since each character echoes as the user types it, output from the previous command may appear on the terminal jumbled with the command being typed ahead. Type-ahead is limited to the size of the terminal input buffer. Default is 192 characters.

### The ER! Prompt

The ER! prompt indicates that PRIMOS was unable to execute the most recent command, for one reason or another, and that PRIMOS is ready to accept another command from the user. The ER! prompt usually is preceded by one or more error messages indicating what PRIMOS thought the trouble was.

Common errors include:

- Typographical errors
- Omitting a password
- Being in the wrong directory
- Forgetting an argument to a command

### Changing the Prompt Message

Users can change the prompt messages displayed at their terminals (both "OK," and "ER!") by using the RDY command. See Chapter 17, CUSTOMIZING YOUR ENVIRONMENT, for details.

## THE NOTION OF DEFAULT

A "default" character, option, value, or action is one that is selected automatically by PRIMOS without your having to specify it explicitly.



For example, the "default" prompt message is "OK,". PRIMOS is set to give you this prompt automatically. You can usually change a default condition to another of your own choosing. For example, you may use the RDY command (explained in Chapter 17) to change the "OK," prompt message.

## TERMINAL KEYBOARD

Most of the user's interaction with PRIMOS takes place at a computer terminal. Here we review the standard functioning of terminals and present certain aspects unique to Prime.

### Basic Layout

The exact layout of the terminal keyboard varies with the type of terminal. Figure 2-8 shows a typical keyboard.

f <sub>1</sub>	f <sub>2</sub>	f <sub>3</sub>	f <sub>4</sub>	f <sub>5</sub>	f <sub>6</sub>	f <sub>7</sub>	f <sub>8</sub>	f <sub>9</sub>	f <sub>10</sub>	f <sub>11</sub>	f <sub>12</sub>	f <sub>13</sub>	f <sub>14</sub>	f <sub>15</sub>	f <sub>16</sub>	D LINE	I LINE	D CHAR	I CHAR
SEND	LOCAL	A-SEND	E-AUX	AUX-ON	PAGE	M-LOCK	FORMS	A-SET	RESET	PRINT	SCRL I	SCRL I	B-TAB	←	↓	HOME	↑	→	TAB
!	"	#	\$	%	&	'	(	)		@	=	~	:	BACK SPACE	7	8	9	CE	
1	2	3	4	5	6	7	8	9	0										
ESC	Q	W	E	R	T	Y	U	I	O	P	LF	RETURN	DEL	BREAK	4	5	6	-	
LOCK	A	S	D	F	G	H	J	K	L	+	*		{	}	1	2	3	ENTER	
										;	:								
SHIFT	Z	X	C	V	B	N	M	<	>	?	/	SHIFT	EOP CLEAR EOF	CLEAR	0	.			
CONTROL										CONTROL									

Typical Terminal Keyboard

Figure 2-8

Besides the usual letter, number, and punctuation symbols, the terminal keyboard also has a variety of special symbols and keys. The number and letter keys are arranged in the same positions as on a standard typewriter. The punctuation marks, however, may be located on different keys.



Special keys fall into the following categories:

- Terminal controls and switches
- Special keys
- Special characters

### Terminal Controls and Switches

Terminal controls and switches affect the ways a specific terminal performs. Depending on what terminal model is available, these may be on the front, side or bottom of the terminal, or beside the standard keyboard.

The controls and switches of importance are:

ON/OFF: This is the power switch. Some terminals have an indicator light which glows when power is on. On some terminal models, this switch may be located at the rear or on the bottom.

LINE/LOCAL: This switch controls whether or not the terminal is sending input to the computer. In LINE Mode, the terminal and the computer are connected; in LOCAL Mode, the terminal acts like a specialized typewriter. This switch is often labeled: ON-LINE/OFF-LINE, REMOTE/LOCAL or LINE (with an indicator light which is ON in LINE Mode, OFF in LOCAL Mode).

UPPERCASE/LOWERCASE: Unlike the SHIFT key, the UPPERCASE/LOWERCASE key affects the meanings of the letter keys only. UPPERCASE causes all letters to print in uppercase. LOWERCASE allows selection between uppercase and lowercase in the standard manner (by using the shift key). On some terminals this is the LOCK key.

On some terminals, this switch is located on the bottom, instead of on the keyboard. This key is often labeled: CASE, UPPER/LOWER or U/C (for Uppercase -- when switched on). Terminals which produce uppercase letters only do not have this switch.

### SPECIAL TERMINAL KEYS

#### • CONTROL

The key labeled CONTROL (or CTRL) changes the meaning of alphabetic and some punctuation keys. Holding down CONTROL while pressing an alphabetic key (or some special keys) generates a "control character".



Control characters do not print. Some of them have special meanings to the computer. (See CONTROL-P, CONTROL-Q, and CONTROL-S, below.)

- RUBOUT or DELETE/DEL

The key labeled RUBOUT has a special use in Prime's text processing utility, RUNOFF, and also in Prime's Office Automation System (OAS). It is not generally meaningful to other standard Prime software. On some terminals it is labeled DELETE or DEL.

- RETURN

The RETURN key ends a line. PRIMOS modifies the line according to any erase or kill characters, and either processes the line as a PRIMOS command, or passes it to a utility such as the EDITOR. RETURN is also called CR, CARRIAGE-RETURN, or NEW-LINE.

- { BREAK  
ATTN } See CONTROL-P  
INTRPT

### SPECIAL CHARACTERS

- Caret (^)

The caret is the default character used by the EDITOR to enter octal numbers and for literal insertion of special characters. On some terminals and printers, the caret prints as an up-arrow (↑).

- Backslash (\)

The backslash is the default EDITOR tab character.

- Double-quote (")

The double-quote character is the default erase character for PRIMOS and all subsystems. Each double-quote erases a character from the current line. Erasure is from right (the most recent character) to left. Two double-quotes erase two characters, three erase three, and so forth. You cannot erase beyond the beginning of a line. The PRIMOS command TERM (described later in this section) allows the user to choose a different erase character, such as BACKSPACE.



- Question mark (?)

The question mark is the default kill character for PRIMOS and all subsystems. Each question mark deletes all previous characters on the line. The PRIMOS command TERM allows the user to choose a different kill character.

- CONTROL-P

The CONTROL-P character instructs PRIMOS to quit immediately (interrupt/terminate) from execution of current command and return to PRIMOS level. The character echoes as QUIT. CONTROL-P is used to escape from undesired activity. It will leave used files open in certain circumstances. Using CONTROL-P is equivalent to hitting the BREAK key, if BREAK is enabled on your terminal.

- CONTROL-S

The CONTROL-S character halts output to the terminal, so that the user may inspect it. A program will run until the output buffer is full; then it will be suspended. Any commands other than CONTROL-S or CONTROL-Q will be placed in the input buffer (until that buffer is full). They will not execute until the suspended program has terminated. Input will not be echoed at the terminal until either CONTROL-P (QUIT) or CONTROL-Q (Continue) is given. This special function is activated by the command TERM -XOFF.

- CONTROL-Q

The CONTROL-Q character resumes output to terminal following a CONTROL-S (if TERM -XOFF is in effect).

- UNDERSCORE ( \_ )

On some devices, the underscore prints as a backarrow (←).

- RESERVED CHARACTERS

The following keyboard characters are reserved by PRIMOS for special uses. They may not be used in filenames:

, ( ) { } [ ] < > ! % ' = + ` @ ~ : | ; ? " \ ^  
rubout/delete space

- SEMICOLON (;)

The semicolon is used as a command separator. Using the semicolon, you can place multiple commands on a single line.



SETTING TERMINAL CHARACTERISTICS

Terminal characteristics may be set with the TERM command. These characteristics remain in effect until you reset them or until you log out. The commonly used TERM options are listed below. Typing TERM with no options returns the full list of TERM options available. The format is:

## TERM options

The common options are:

<u>Option</u>	<u>Function</u>
-ERASE character	Sets user's choice of erase character in place of the " default.
-KILL character	Sets user's choice of kill character in place of ? default.
-BREAK { ON OFF }	Enables or disables use of CONTROL-P as a break character, to interrupt a running program or command. Default, enabled at LOGIN, is BREAK ON.
-XOFF	Enables X-OFF/X-ON feature, which allows users to suspend terminal output temporarily and to resume it at the point of suspension. Output is halted by typing CONTROL-S and is resumed by typing CONTROL-Q. Also sets terminal to full duplex (default value).
-NOXOFF	Disables X-OFF/X-ON feature (default).
-DISPLAY	Returns list of currently set TERM characters. Also displays current duplex, break, and X-ON/X-OFF status.

Other TERM options are discussed in the PRIMOS Commands Reference Guide.

Sending Messages From Your Terminal

You may communicate with users at other terminals by using the MESSAGE command.







# 3

## Getting Started with PRIMOS

### INTRODUCTION

This user's guide contains those PRIMOS commands that will be of use to most programmers. Depending upon your application, there are many other PRIMOS commands that may simplify your task or increase efficiency. In this chapter we introduce the essential commands so that you can begin working on the system.

### Using PRIMOS

PRIMOS recognizes more than 100 commands. Some of these commands invoke subsystems that themselves respond to subcommands or extensive dialogues. However, most users can do the majority of their program development using about a dozen commands. This section introduces the essential commands needed by all users. These commands allow you to:

- Gain admittance to the computer system (LOGIN)
- Connect to another directory (ATTACH)
- Return to the origin directory (ORIGIN)
- Change your login password (CHANGE\_PASSWORD)
- Create new directories for work organization (CREATE)
- Examine files (SLIST)



- Examine the current directory and its contents (LD)
- Rename file system objects (CNAME)
- Copy file system objects (COPY)
- Remove unwanted file system objects (DELETE)
- Protect file system objects from accidental deletion (SET\_DELETE)
- Request information on individual PRIMOS commands, groups of commands, or general topics (HELP)
- Record everything you do during terminal sessions (COMOUTPUT)
- Complete a work session (LOGOUT)

Table 3-1 summarizes these commands.

## ACCESSING THE SYSTEM

### Obtaining a User Id

Before you can use the system, you must register with your System Administrator (a person) to obtain a user id and perhaps a login password (if your system uses login passwords). A user id is a name up to 32 characters long. The first character must be a letter; the rest may be letters, digits, or the special characters ".", "\_", and "\$". A login password is a string up to sixteen characters long. It may contain any ASCII character (listed in Appendix C) except CR, CONTROL-P, CONTROL-S, CONTROL-Q, and PRIMOS reserved characters (listed in Chapter 2).

### Logging In

Once you have your user id, you are ready to work on the system. You begin work by "logging in". "Logging in" identifies you to the system and establishes the initial contact between you and the system via a terminal. When you finish logging in, you are attached to your origin directory; your access rights on the system have been established; and PRIMOS is ready to execute other commands for you.

In its briefest format the LOGIN command is simply:

LOGIN

The system then requests a user id with the prompt:

User id?



Table 3-1  
Essential PRIMOS Commands

Command	Function
ATTACH	Connects you to the directory specified
CHANGE_PASSWORD	Changes your login password
CNAME	Changes the name of a file system object
COMOUTPUT	Records terminal sessions
COPY	Makes a copy of a file system object
CREATE	Creates an empty sub-directory in the current directory
DELETE	Removes unwanted file system objects
HELP	Provides information on PRIMOS commands
LD	Lists the contents of the current directory
LOGIN	Admits you to the system and attaches you to your origin directory
LOGOUT	Disconnects you from the system
ORIGIN	Reattaches you to your origin directory
SET_DELETE	Protects a file system object from accidental deletion
SLIST	Lists the contents of a file



You must type in your user id. If you have a login password, your password will now be requested by the prompt:

Password?

You must type in your login password. For security reasons, the password will not appear on the screen as you type it.

Some systems organize users into specific groups called projects. If your system uses the project structure, PRIMOS may now request a project id with the prompt:

Project id?

You must type in your project id. If you receive a project id prompt and you do not have a project id, see your administrator.

You must always provide the system with your user id; whether you must supply a password or project id depends upon your installation. Once you have successfully provided all of the information requested, the login procedure is completed, and PRIMOS will respond as in the following example:

LOGIN

User id? JONES

Password? NIX [Password entered will not appear on the terminal.]

Project id? RESEARCH

JONES (user 64) logged in Friday, 23 Apr 82 11:23:28.

Welcome to PRIMOS version 19.0.

Last login Friday, 23 Apr 82 09:49:44.

The word NIX, in this example, is the login password. RESEARCH is the project id. The number in parentheses is the PRIMOS-assigned user number. The time is expressed in 24-hour format (hh:mm:ss).

If you misspell your user id or password, you will receive the message:

Invalid user id or password; please try again.

If you enter a project id incorrectly, you will receive the message:

Invalid project id; please try again.

If you repeat the login process and still have trouble, ask your administrator for help. If the system itself is fully loaded, a message such as "maximum number of users exceeded" may be displayed. In this case, try to log in again later, when some other user may have logged out.



Alternate Form of the LOGIN Command: You may also enter arguments and options on the same line as the LOGIN command (although for security reasons the System Administrator for your installation may disallow passwords on the login line). The format is:

LOGIN [user-id [login-password][-ON system-name]] [-PROJECT project-id]

<u>Argument/Option</u>	<u>Description</u>
user-id	Your user identification name.
login-password	May be included on login line if your system allows.
-ON system-name	Names the system used for remote login across PRIMENET network. Remote login is explained in Chapter 14.
-PROJECT project-id	Associates the user with a particular project.

If you give a login password or -ON option on the command line, you must also give your user id there. For example:

LOGIN JONES NIX -ON NODE3 -PROJECT RESEARCH  
[PRIMENET 19.0 NODE3]

JONES (user 64) logged in Friday, 23 Apr 82 11:23:28.  
Welcome to PRIMOS version 19.0.  
Last login Friday, 23 Apr 82 09:49:44.

### The Origin Directory

Once you have successfully logged in, PRIMOS will connect you to a directory set by the System Administrator as your initial attach point. This directory is your "origin directory" and is now also your "current directory."

#### Note

In order to use fully the commands described below, you need to be included in an access control list (ACL) that grants you access rights for file system objects in your origin directory. Full access rights for directories are protect, delete, add, list, and use. Full access rights for files and segment directories are read and write. You may or may not have full rights depending upon how your user id has been defined. If you are attached to your origin directory and are unable to execute commands, you may not have the necessary rights in an



ACL. See your administrator. This chapter assumes you are using the ACL protection system unless otherwise noted. ACLs are explained fully in Chapter 16.

### Moving From the Current Directory

You can move to another directory in the PRIMOS tree structure with the ATTACH command, provided that you have access rights to the new directory. The format is:

ATTACH new-directory

new-directory is the pathname of the directory that will become the new current directory.

Access Requirements: In order to attach to an ACL-protected directory, you must have use (U) access to it.

To attach to a passworded directory, you must supply any necessary passwords, as in:

A 'BEECH SECRET>BRANCH5'

(If you need information on passworded directories, consult Appendix F.)

Possible Attaching Errors: If an attempt to ATTACH to a subdirectory fails because the directory is not found (as, for example, when a directory name is misspelled), PRIMOS returns the error message:

Not found. directory-name (ATTACH)

An attempt to ATTACH to a top-level directory may fail because the directory does not exist, because PRIMOS cannot reach the system where it does exist, or because you do not have the right to attach to it. In any of these cases, PRIMOS returns the error message:

Top-level directory not found or inaccessible. directory-name (ATTACH)

If the ATTACH command fails because a subdirectory is inaccessible through ACL protection, PRIMOS returns the error message:

Insufficient access rights. directory-name (ATTACH)

If an incorrect password is given with the ATTACH command for directories that require passwords (explained in Appendix F), PRIMOS returns the error message:

Bad Password. directory-name (df\_unit\_)



In all of these cases, the user remains attached to the previous current directory.

### Returning to the Origin Directory

You may return to your origin directory at any time by using the command:

ORIGIN

### Assigning or Changing Your Login Password

Once logged in, you may assign yourself a login password (if you do not have one) or change your current login password with the CHANGE\_PASSWORD command. A login password may be up to 16 characters in length. It may contain any ASCII character except PRIMOS reserved characters (listed in Chapter 2). The format for assigning a new login password is:

```
{ CHANGE_PASSWORD }
{ CPW }
```

The format for changing an existing login password is:

```
{ CHANGE_PASSWORD } old-login-password
{ CPW }
```

Following either of the two commands above, the system will request the new login password with the prompt:

New password?

As usual, the new password will not appear on the terminal as you type it. The system will request the new login password a second time, for verification, with the prompt:

Reenter new password for confirmation:

PRIMOS will print an appropriate error message if the old password is incorrect, if the two new passwords entered do not match, or if the format of the new password is illegal. In any of these cases, the old password is not changed.

An example of CHANGE\_PASSWORD follows:

CPW NIX

New password? STYX [Password entered will not appear on terminal.]

Reenter new password for confirmation: STYX [Again, nothing visible]

OK,



If you wish to remove an existing login password without substituting another, use the carriage return when the new-password prompts appear.

## CREATING NEW FILES AND DIRECTORIES

### Creating Files

Text files are created (and modified) using the line text editor (ED). They are printed on the line printer using the SPOOL command. The ED and SPOOL processes are discussed in Chapter 4.

### Creating Directories

To organize tasks and work efficiently, you can create new sub-UFDs, also called subdirectories. These sub-UFDs can be created within UFDs or other sub-UFDs with the CREATE command. They can contain files and/or other subdirectories. The format is:

CREATE pathname

pathname may be:

- The name of a new subdirectory to be created within the current directory.
- The pathname of a new subdirectory to be created within some other directory.

Access: You must have Add (A) access rights to create a new sub-UFD.

For example:

```
ATTACH BEECH
CREATE BRANCH6
```

creates the subdirectory BRANCH6 in the directory BEECH.

```
CREATE ELM>BRANCH1
```

creates the subdirectory BRANCH1 in the UFD ELM.

Two files or sub-UFDs of the same name are not permitted in a directory. If this is inadvertently attempted, PRIMOS will return the message:

```
Already exists. DIRECTORY-NAME (CREATE)
ER!
```



EXAMINING THE CONTENTS OF FILES AND DIRECTORIESExamining File Contents

You can examine the contents of any text file at the terminal with the SLIST command. The format is:

SLIST pathname

The file specified by the given pathname is displayed at the terminal. You can stop the terminal display temporarily for inspection if you previously used the TERM -XOFF command, discussed under SETTING TERMINAL CHARACTERISTICS in Chapter 2.

Access: You must have read (R) access to display a file.

Examining Directory Contents

After logging in or attaching to a directory, you can examine the contents of the directory with the LD command. In its simplest form, the format is:

LD

In this form, the LD command prints a list of all items in the current directory, four to a line and sorted alphabetically within type. Types are listed in the order: file, segment directory, directory, access category.

Access: In order to list a directory, you must have list (L) access to the directory.



For example, suppose that the current directory is called SMITH. The following list will be generated when LD is entered at the terminal:

OK, LD

<DISK>SMITH (LUR), Records= 32, Quota= 163 /500

Files= 7.

ABBREV  
LETTER

BUDGET1  
PAYROLL.INFO

BUDGET2  
REPORT

MAIL

Segment Directories= 3.

EXAMPLE.SEG

PROGRAM.SEG

TABULATION.SEG

Directories= 2.

SUB1

SUB2

Access Categories= 2.

GENERAL.ACAT

LETTER.ACAT

The letters in parentheses following the pathname indicate the access rights for the directory available to the current user. In this example, the user has list (L), use (U), and read (R) access rights. If the directory is not protected by access control lists (ACLs), the words "Owner" or "Non-owner", explained in Appendix F, will appear within the parentheses. The number following "Records=" is the number of decimal disk records used by the directory itself and all of its files and segment directories (here 32). The numbers following "Quota=" are, respectively, the number of records used by the directory and its entire subtree (here 163), followed by the maximum number of records permitted for use by the directory and its subtree (here 500). If the second number were 0, there would be no maximum other than the limit of the disk. Quotas are discussed further in Chapter 17.

Access: You must have list (L) access in order to list a directory. If you attempt to use the LD command on a directory to which you have use but not list access, you will receive the message:

No information. (current-directory) (ld)

Options for the LD Command: The LD command allows many additional instructions, called options, which a user may specify on the same command line. These options may request additional information about directory entries, such as user access rights, date and time created, size, and protection settings. They may also be used to select information for part of the directory. For example, they may request information only about files, or even about certain groups of files. These options are discussed in PRIMOS Commands Reference Guide.



CHANGING THE NAMES OF FILE SYSTEM OBJECTS

It is often convenient or necessary to change the name of a file system object (file, subdirectory, segment directory, access category). This is done with the CNAME command. This format is:

CNAME old-name new-name

old-name is the pathname of the object to be renamed, and new-name is the new objectname of that object. For example:

CN REPORTS>DRAFT FIRSTDRAFT

The entry named DRAFT in the UFD REPORTS is changed to FIRSTDRAFT. Since no disk was specified, all MFDs, starting with logical disk 0, are searched for the UFD with the name REPORTS.

If new-name, in this example FIRSTDRAFT, already exists, PRIMOS will display the message:

Already exists. FIRSTDRAFT (CNAME)  
ER!

A misspelled old-name, such as DRIFT instead of DRAFT, prompts the "Not found" message:

CN REPORTS>DRIFT FIRSTDRAFT  
Not found. DRIFT (CNAME)  
ER!

Access: You must have delete (D) and add (A) access to use the CNAME command.

COPYING FILE SYSTEM OBJECTS

You may wish to place a copy of an object in another directory or to generate a work copy for test revisions, while still retaining the original version. The COPY command copies files, directories, segment directories, and access categories either within the same directory or from one directory to another. In its simplest form, the format is:

COPY pathname [new-pathname]

pathname gives the name of the object to be copied. The object itself is not removed from its directory or altered in any way.

new-pathname gives the pathname of the newly copied object. If new-pathname is not specified, the object is copied into the current directory under its original name.



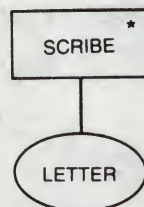
Access: In order to copy an object in an ACL-protected system, you must have read (R) access rights to files and segment directories being copied and use (U) access to the parent directory. You must also have add (A) and use (U) access rights in the directory containing new-pathname. If an object with the new name already exists in the target directory, you must have delete (D) access as well as add (A), so that the new entry may overwrite the old.

### Examples

Suppose LETTER is a text file in the current directory SCRIBE. The command:

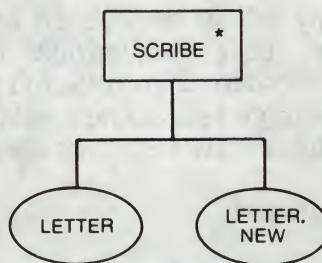
`COPY LETTER LETTER.NEW`

copies the file LETTER into a new file called LETTER.NEW in the current directory. The original file LETTER remains in the directory. Figures 3-1 and 3-2 illustrate the directory SCRIBE before and after the COPY command is given.



\* = attach point

Directory Before COPY Command Is Given  
Figure 3-1

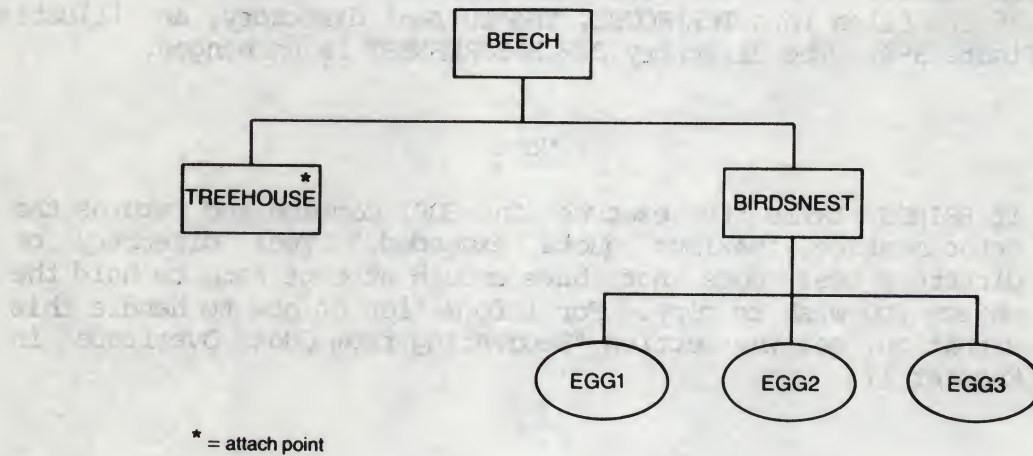


\* = attach point

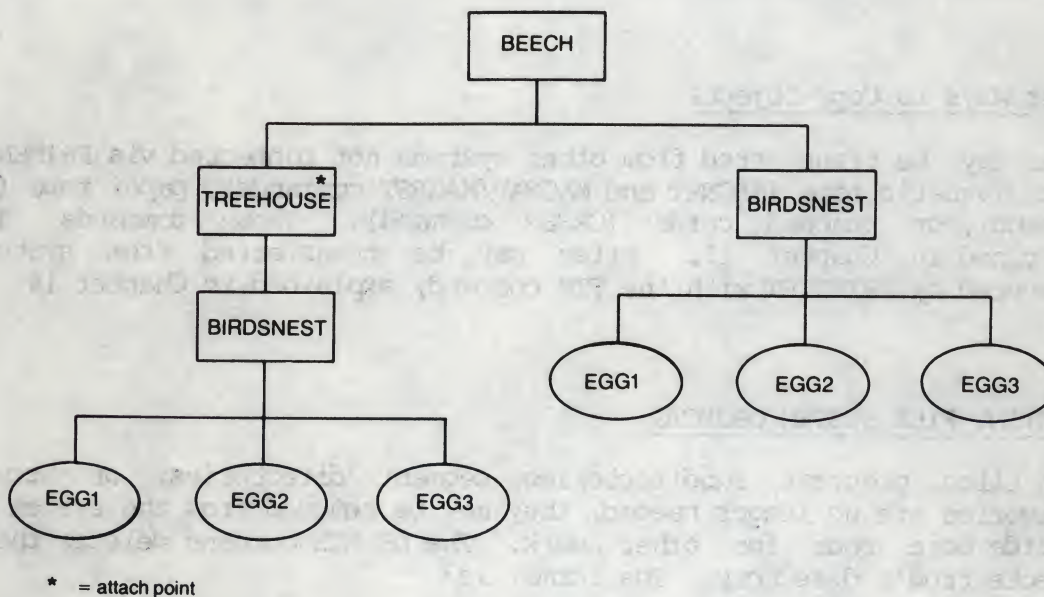
Directory After COPY Command Is Given  
Figure 3-2



As a second example, suppose you are attached to the subdirectory **TREEHOUSE** in the UFD **BEECH**. **BIRDSNEST** is also a subdirectory in **BEECH** and contains the files **EGG1**, **EGG2**, and **EGG3**. You wish to copy **BIRDSNEST** and all its files into **TREEHOUSE**. Figures 3-3 and 3-4 illustrate the directory **BEECH** before and after the **COPY** command is given.



Directory Before COPY Command Is Given  
Figure 3-3



Directory After COPY Command Is Given  
Figure 3-4



The command interaction is as follows. The command:

```
COPY BEECH>BIRDSNEST
```

first queries you for permission to copy the directory:

```
OK to copy directory "BEECH>BIRDSNEST" to "BIRDSNEST"?
```

If you answer Y or Yes, COPY will copy the subdirectory BIRDSNEST and all of its files into TREEHOUSE, the current directory, as illustrated in Figure 3-4. The directory BEECH>BIRDSNEST is unchanged.

#### Note

If PRIMOS fails to execute the COPY command and returns the error message "Maximum quota exceeded," your directory or directory tree does not have enough storage room to hold the object you wish to copy. For information on how to handle this situation, see the section "Recovering from Quota Overloads" in Chapter 17.

#### Options for the COPY Command

The COPY command allows many additional instructions, called options, which a user may specify on the same command line with COPY itself. These options modify the copy process. The options are discussed in PRIMOS Commands Reference Guide.

#### Other Ways to Copy Objects

Files may be transferred from other systems not connected via PRIMENET using magnetic tape (MAGNET and MAGSAV/MAGRST commands), paper tape (ED command), or punched cards (CRSER command). These commands are described in Chapter 13. Files may be transferred from systems connected by PRIMENET with the FTR command, explained in Chapter 14.

#### DELETING FILE SYSTEM OBJECTS

When files, programs, subdirectories, segment directories, or access categories are no longer needed, they may be removed from the system to provide more room for other work. The DELETE command deletes these objects from a directory. The format is:

```
DELETE pathname
```



If the object is a directory or an access category, PRIMOS will query the user to double-check that the command is intended for this entry. For example:

```
OK, DELETE SUBDIR5
OK to delete directory "SUBDIR5"? yes
OK,
```

If the object specified is not in the directory to which you are attached, PRIMOS returns a "not found" message. For example:

```
OK, DELETE BRANCH23
Not found. "BRANCH23" (delete)
```

Access: You must have delete (D) access to the directory in order to use the DELETE command. (Remember that the LD command lists your access rights on a directory in parentheses in the first line of the display it gives you.)

#### Options for the DELETE Command

The DELETE command, like LD and COPY, also allows several options, which tell PRIMOS how to proceed with deletion activities. These options are discussed in PRIMOS Commands Reference Guide.

#### PROTECTING FILE SYSTEM OBJECTS FROM ACCIDENTAL DELETION

You can prevent accidental deletion of a file, segment directory, or directory with the SET\_DELETE command. When this protection exists on an object, PRIMOS will query you before deleting the object. SET\_DELETE works only for objects in directories protected by ACLs. You must have delete (D) access to the directory containing the object in order to use the SET\_DELETE command on the object. The format is:

```
SET_DELETE pathname [-PROTECT •]
                    [-NO_PROTECT]
```

<u>Argument/Option</u>	<u>Description</u>
pathname	Names the object to be delete-protected.
{ -PROTECT -PRO }	Provides a query to the user before deleting the object.
{ -NO_PROTECT -NPRO }	Removes any provision for a query before deleting the object. This is the default state, in which an object is created.



If you give the SET\_DELETE command without either option, -PROTECT is assumed and the object specified is delete-protected.

In the example below, the file IMPORTANT is delete-protected. The DELETE command will query you before deleting the object:

```
OK, SET_DELETE IMPORTANT -PROTECT
OK, DELETE IMPORTANT
"IMPORTANT" protected, ok to force delete? no
File is delete-protected. Unable to delete "IMPORTANT" (delete)
OK,
```

The same querying occurs if you try to delete a directory that contains protected objects (even though the directory itself is not delete-protected). For example, assume the subdirectory CRUCIAL is contained within the directory PAPERS. You are attached to PAPERS:

```
OK, SET_DELETE CRUCIAL
OK, ATTACH MYUFD
OK, DELETE MYUFD>PAPERS
"Ok to delete directory "MYUFD>PAPERS? YES
"MYUFD>PAPERS>CRUCIAL" protected, Ok to force delete? YES
OK,
```

If you had answered "NO" to the last query, you would have received the following response:

The directory is not empty. Unable to delete "PAPERS" (delete)

Determining Whether Objects Are Delete-protected: To see whether an object is delete-protected, use the LD -DETAIL or LD -PROTECT command. If the object is delete-protected, the letters "pr" will appear on the information line for the object. For example, suppose you wish to examine the delete protection on objects in the UFD JONES, to which you are attached:

```
OK, LD -PROTECT
<DISK>JONES (ALL), Records= 10, Quota= 10 / 500

Files= 2.
```

```
ALL sam      pr IMPORTANT
ALL sam      TRIVIAL
```

In this example, the file IMPORTANT is delete-protected, but the file TRIVIAL is not.



CONTROLLING ACCESS TO FILES AND DIRECTORIES

You may wish to secure your files and directories against unauthorized viewing or changes. A protection system exists for this purpose and consists of access control lists (ACLs). This system and its related commands are described in Chapter 15. An older, alternate protection system also exists through directory passwords. Password protection is described in Appendix F.

REQUESTING INFORMATION WITH THE HELP COMMAND

You may be working at a terminal and wish to refresh your memory quickly on the syntax of a command or on the details of its function. PRIMOS itself maintains information on individual PRIMOS commands, groups of commands, and general topics. You may obtain this information with the HELP command. Its format is:

HELP [name]

name is the name of the command or topic on which you wish information. name may be a command abbreviation. If name is omitted, a list of available commands and topics is printed with a descriptive phrase for each. At the end of each discussion are any references to further information and the date that the HELP information was written or updated.

For example:

OK, HELP ORIGIN  
ORIGIN

Return to origin directory

Abbreviation: OR

ORIGIN

The ORIGIN command changes the user's attach point to his origin directory. It takes no arguments.

For further information, see the Prime User's Guide.

May 1981  
OK,

RECORDING TERMINAL SESSIONS

Sometimes you may wish to record your interactive sessions, especially if you use a screen terminal where the history of your work would otherwise be lost. Recording a session can be particularly helpful if you are a new user, or if something puzzling is happening about which you want someone's advice.



When you give the command:

COMOUTPUT pathname

PRIMOS creates a file named pathname and records in it whatever you type at the terminal and whatever responses the system makes to your commands. The commands and responses continue to appear on your terminal, as well.

To stop recording material and close the file, give the command:

COMOUTPUT -END

Once you have closed the file, you can read it with the SLIST command, or you can request a printed copy with the command:

SPOOL pathname

The SPOOL command is explained fully in Chapter 4.

The following example illustrates how to form a COMOUTPUT (command output) file:

```
OK, COMOUTPUT SESSION.COMO
OK, DATE
16 Apr 82 14:22:52 Friday
OK, COMO -END
OK,
```

A file named SESSION.COMO now exists in the current directory and contains the record of all input to and output from PRIMOS following the COMOUTPUT SESSION.COMO command, up to and including the COMO -END command. Listing SESSION.COMO with the SLIST command reveals the following contents:

```
OK, SLIST SESSION.COMO
OK, DATE
16 Apr 82 14:22:52 Friday
OK, COMO -END
OK,
```

The COMO -END command is part of the recorded material but the COMOUTPUT SESSION.COMO command is not.

#### Note

If a file named pathname already exists, it is usually overwritten by the COMOUTPUT command. However, you can specify that the new material be placed at the end of an existing file. For details on this, and on other extensions of the COMOUTPUT command, see Chapter 10.



Access: You must have add (A) access to use the COMOUTPUT command, or write (W) access if you are overwriting an existing file.

### SYSTEM INFORMATION

PRIMOS supports many commands that provide useful information about both the user and the system. Appendix G tabulates the information available and the commands used to obtain it.

### STOPPING UNWANTED COMMANDS

You can interrupt a running program or command by using the control character:

#### CONTROL-P

To use CONTROL-P, depress the CONTROL key. While CONTROL is still depressed, depress the character P as well. Your terminal activity will stop. You will receive the notification "QUIT", followed by the "OK," prompt. You are now at PRIMOS command level and may enter your next command.

For example:

OK, LD

<DISK>JUDY (ALL), Records= 91, Quota= 135 / 500

Files= 39.

FILE1  
FILE5  
FILE9  
(CONTROL-P)  
QUIT.  
OK,

FILE2	FILE3	FILE4
FILE6	FILE7	FILE8

#### Note

Interrupting programs and commands can leave files open and may lead to unwanted results in the behavior of subsequent commands. Therefore, whenever you use CONTROL-P, you should follow it with the commands:

CLOSE -ALL; RLS

These commands close any files the interrupted command(s) may have left open and free resources for future use.



COMPLETING A WORK SESSION

When finished with a session at the terminal, give the LOGOUT command. The format is:

LOGOUT

PRIMOS acknowledges the command with the following message:

user-id (user number) logged out weekday, date time.  
Time used: xxh xxm connect, xxm xxs CPU, xxm xxs I/O

The various parts of this display have the following meaning:

<u>Term(s)</u>	<u>Description</u>
number	The number assigned at LOGIN
time	Expressed in 24-hour format (hh:mm:ss)
xxh xxm connect	The amount of elapsed clock time between LOGIN and LOGOUT in hours and minutes
xxm xxs CPU	Central Processing Unit time consumed in minutes and seconds
xxm xxs I/O	The amount of input/output time used in minutes and seconds

For example:

LO

JANE (user 42) logged out Tuesday, 28 Apr 81 13:52:20.  
Time used: 01h 22m connect, 01m 14s CPU, 03m 35s I/O.

It is good practice to log out after every session. This closes all files and releases the PRIMOS process to another user. However, if you forget to log out, there is no serious harm done. The system will automatically log out an unused terminal after a time delay. This delay is set by the System Administrator. The default is 1000 minutes, but most System Administrators will lower this value.



# 4

## Creating and Listing Files

### ENTERING AND MODIFYING PROGRAMS — THE EDITOR

Programs are normally entered into the computer using Prime's Text EDITOR (ED). This EDITOR is a line-oriented text processor. That is, it enters and modifies text on a line-by-line basis, keeping track of its current location by a line pointer that is always located at the last line processed (whether the processing action is printing, locating, moving the pointer, and so on). The EDITOR operates in two modes: INPUT and EDIT.

#### Using the EDITOR

When creating a new file, the EDITOR is invoked by:

```
ED
```

which places the EDITOR in the INPUT mode. When modifying an existing file, the EDITOR is invoked by:

```
ED filename
```

which places the EDITOR in the EDIT mode.

A RETURN with no preceding characters on that line switches the EDITOR from one mode to another.



INPUT Mode

The INPUT mode is used when entering text information into a file (for example, creating a program). The word INPUT is displayed at the user's terminal to indicate that the EDITOR has entered that mode. The RETURN key terminates the current line and prepares the EDITOR to receive a new line.

Tabulation is done with the backslash (\) character. Each backslash moves the text which follows it to the next tab setting on the line. The default tabs are at columns 6, 15, and 30. These settings may be overridden and up to 8 tab settings may be specified by the user with the TABSET command. (TABSET will be described later.)

A RETURN without any preceding characters puts the EDITOR into the EDIT mode.

EDIT Mode

The EDIT mode is used when the contents of the file are to be modified. More than 50 commands are available, although users will find that a small subset of these will suffice for most purposes. The commands are listed and described in detail later in this section.

In EDIT mode, the EDITOR maintains an internal line pointer at the current line (the last line processed). Commands such as TOP, BOTTOM, FIND, and LOCATE, move this pointer. WHERE prints out the current line number; POINT moves the pointer to a specified line number. The MODE NUMBER command causes the line number to be printed out whenever a line of text is printed. All commands for location and modification begin processing with the current line.

A RETURN without any preceding characters puts the EDITOR into the INPUT mode.

Special Characters

In either mode, a single character can be erased with the erase character. The default erase character is a double quotation mark ("). For each " typed, a character is erased (from right to left). The entire current line may be deleted by typing the kill character. The default kill character is a question mark (?). A line followed by a ? is null, and a RETURN at that point will switch the EDITOR into the other mode.

In INPUT mode, the semicolon (;) is equivalent to a CR (ends a line of input). In EDIT mode, semicolons in a character string are treated as a printing character, as in the strings that follow a CHANGE command. Otherwise, in EDIT mode, semicolons act as line terminators to separate multiple commands entered on the same line.



A special character, such as a question mark or semicolon, may be entered literally in either mode by preceding it with the escape character, a caret (^). To enter carets literally, type two carets. The result displays as two carets on the terminal, but prints as one caret on the printer and is interpreted as a single caret by compilers. On some terminals and printers, the caret prints as an up-arrow (↑).

Special characters may be changed using the TERM command (explained in Chapter 2). From within the EDITOR, special characters may be changed with the SYMBOL command, described fully in the New User's Guide to Editor and Runoff.

### Saving Files

An EDITOR session is ended from EDIT mode. The command:

FILE filename

writes the current version of the edited file to the disk under the name filename. The specified file will be created if it did not previously exist or overwritten if it does exist. If an existing file is being modified, the command:

FILE

writes the edited version to the disk with the old filename. After execution of the filing command, control is returned to PRIMOS.

A file may also be saved without leaving the EDITOR. The command:

SAVE filename

saves a file in its current state under filename. The filename may be the original name of the file or a new name. You may then resume editing your current file. If filename is not specified, the current filename is used. The name of the file is then printed on the terminal screen.

### Useful Techniques

The following will aid you in using Prime's EDITOR:

Tab Settings: When entering source code, you can save time by using the TABSET command. In INPUT mode, each backslash (\) character is interpreted as one tab setting; the default values are columns 6, 15, and 30. Tabs may be set to whatever values each programmer finds useful. Setting a tab near column 45 makes entry of in-line comments simple. The use of such comments in programs is strongly advised.



Moving Lines of Code: Any number of lines can be moved from one location to another using the DUNLOAD command. DUNLOAD deletes these lines as it writes them into an auxiliary file. A LOAD command loads the new file at the desired point. Any number of lines can be copied from one location in a program to another by using the UNLOAD command. UNLOAD does not delete the lines as it writes them into an auxiliary file. A LOAD command loads the copy from the new file at the desired point.

Overlaying Comments After Code is Written: Comments may be easily added to an existing source program with the OVERLAY command in conjunction with the TABSET command.

Finding a Line by Label or Statement Number: The FIND command may be used to locate a statement number in a FORTRAN program or a label in a COBOL or PL/I program.

Modifying a Line Without Changing Character Positions: The MODIFY command is used when a line must be modified but the absolute column alignment must remain the same.

Column Display: Entering source code and other data is facilitated by the column display feature. A banner of column numbers is displayed on the terminal for an alignment guide. The MODE COLUMN command, given in EDIT mode, causes this display to be printed each time INPUT mode is entered during an EDITOR session.

FORTRAN Programs: When entering FORTRAN programs, you may find the TABSET command helpful to reset tabs to columns 7 and 45.

PL/I Programs: The SEMICO special character, automatically set as the semicolon (;), normally signals the EDITOR of the end of a line or command. PL/I programmers will wish to tell the EDITOR to interpret the semicolon as a literal character. You may do this in either of two ways.



The easiest way is to use the `MODE NOSEMI` command. This command deactivates the line-ending function of the semicolon and allows it to be used as a literal character when you are in `INPUT` mode or when you are appending, inserting, or retyping information in `EDIT` mode. For example:

```
OK, ED SHEEP
EDIT
MODE NOSEMI
NEXT
BA, BA BLACK SHEEP
APPEND ; HAVE YOU ANY WOOL
BA, BA BLACK SHEEP; HAVE YOU ANY WOOL
```

The semicolon still terminates (or separates) `EDITOR` commands that do not insert new text into the file. For example:

```
OK, ED SHEEP
EDIT
MODE NOSEMI
TOP; NEXT
BA, BA BLACK SHEEP
```

The `MODE SEMI` command causes the `EDITOR` to recognize the semicolon as the line terminator. This is the default state, and the `MODE SEMI` command returns the `EDITOR` to this state.

A second way to free the semicolon is to use the `SYMBOL` command to change the `SEMICO` special character from the semicolon to something else. For example:

```
SYMBOL SEMICO {
```

In this example, the brace becomes the `EDITOR`'s line-ending symbol, and the semicolon is freed to be used as a regular print character for its `PL/I` functions.

#### EDITOR'S ERROR MESSAGES

In `EDIT` mode, if you give the `EDITOR` a command that it cannot understand, you will receive one of the following error messages:

- `BAD string` — string is a character or characters (usually an abbreviation) that the Editor does not recognize.
- `?` — Your input could not be interpreted as any of the `EDITOR` commands. This is often a result of thinking that you are in `INPUT` mode when you are still in `EDIT` mode.



BASIC EDITOR COMMANDS

You should be able to do most of your text-editing using the following selection of EDITOR commands. Valid command abbreviations are underlined.

- The PRINT Command
- Location Commands
  - TOP
  - NEXT
  - POINT
- String-Finding Commands
  - LOCATE
  - FIND
  - NFIND
  - FIND(n)
  - NFIND(n)
- Text-Changing Commands
  - APPEND
  - CHANGE
  - DELETE
  - INSERT
  - IB
  - RETYPE
  - OOPS
  - DUNLOAD
  - LOAD
  - UNLOAD
- Ending and Saving an EDITOR Session Commands
  - QUIT
  - FILE
  - SAVE

These commands are discussed in detail in the following few pages. All of the EDITOR commands are listed in Appendix E of this guide and in the New User's Guide to EDITOR and RUNOFF.

Note

The string argument in the commands in this section is any series of ASCII characters including leading, trailing, or embedded blanks. A semicolon terminates the command unless it appears within delimiters (as in the CHANGE, MODIFY, or GMODIFY commands) or is preceded by the escape character (^), which enters it literally.



Sample File

The following FORTRAN program is used in all the examples in this section:

C This program generates the numbers 1 to 10  
C and prints the numbers on the terminal screen.  
C

```

      DIMENSION LNUMB (50)
      DO 100 I = 1, 10
        LNUMB (I) = I
        WRITE (1, 200) LNUMB (I)
200   FORMAT (10X, I5)
100   CONTINUE
      CALL EXIT
      END

```

The PRINT Command

The PRINT command prints n lines of your file, including the current line, and makes the last line printed the new current line. The format of the PRINT command is:

PRINT [n]

n is the number of lines you want printed. If n is -1, 0, or omitted, its value is 1. If n is negative, EDITOR moves the pointer back n lines from the current line, and then prints one line, which is the new current line. For example:

PRINT 5

.NULL.

C This program generates the numbers 1 to 10  
C and prints the numbers on the terminal screen.

C

```

      DIMENSION LNUMB (50)

```

PRINT 2

```

      DIMENSION LNUMB (50)

```

```

      DO 100 I = 1, 10

```

PRINT -2

```

      DIMENSION LNUMB (50)

```

The space between PRINT and n is optional. A PRINT immediately after the following commands yields .NULL.: TOP, BOTTOM, DELETE, DUNLOAD, LOAD.

Location Commands

Location commands move the pointer to a specific line. EDITOR's specific location commands are TOP, BOTTOM, NEXT, and POINT.



The TOP Command: The TOP command moves the pointer to the null line at the top of the file, just above the first line of text. The format of the TOP command is:

TOP

For example:

TOP  
PRINT  
.NULL.  
PRINT 2  
.NULL.

C This program generates the numbers 1 to 10

The BOTTOM Command: The BOTTOM command moves the pointer to the bottom of the file, just below the last line of text. The format of the BOTTOM command is:

BOTTOM

For example:

BOTTOM  
PRINT  
.NULL.  
BOTTOM  
PRINT -3  
CALL EXIT  
PRINT 5  
CALL EXIT  
END  
BOTTOM

The NEXT Command: The NEXT command moves the pointer n lines and prints the new current line. Positive values of n move the pointer down towards the bottom of the file; negative values move the pointer up towards the top. The format of the NEXT command is:

NEXT [n]

If n is 0 or unspecified, the default value of 1 is used. If n is great enough to move the pointer beyond the top or bottom null line, the pointer stops at the null line, and either TOP or BOTTOM is printed.



For example:

```

TOP
NEXT
C This program generates the numbers 1 to 10
NEXT 5
      LNUMB (I) = I
BOTTOM
NEXT
BOTTOM

```

The POINT Command: The POINT command positions the pointer at line n. The line numbers are not actually part of your file; EDITOR generates them for its own reference. The format of the POINT command is:

POINT n

The POINT command is equivalent to the sequence TOP, NEXT n. The value of n must be greater than 0. POINT 0 will give you an error message. POINT 1 is equivalent to TOP, NEXT. If n is greater than the number of lines in the file, the pointer will be left at the bottom. For example:

```

POINT 5
DO 100 I = 1, 10
POINT 7
      WRITE (1, 200) LNUMB (I)
POINT -4
BAD POINT
POINT 2
C and prints the numbers on the terminal screen.

```

### String-finding Commands

The LOCATE and FIND commands reposition the pointer to the first line below the current line containing the specified string. The NFIND command repositions the pointer to the first line below the current line which does not begin with the specified string.

These commands distinguish between uppercase and lowercase letters in a specified string. If you are unable to find old lines in your file, but can find newly inserted ones, and your current display is set for all CAPS, the CASE control on your terminal may be in the wrong position.

The LOCATE Command: The LOCATE command locates the first line below the current line which contains string anywhere in that line and prints the line on your terminal. The format of the LOCATE command is:

LOCATE string



If no line containing string is found below the current line, BOTTOM will be printed and the pointer left at the end of the file. The string cannot contain commas. For example:

```

PRINT 5
.NULL.
C This program generates the numbers 1 to 10
C and prints the numbers on the terminal screen.
C
      DIMENSION LNUMB (50)
TOP
LOCATE DIMENSION
      DIMENSION LNUMB (50)

```

The FIND Command: The FIND command is a specialized version of the LOCATE command. It searches for a string and prints the string when found. The string, however, must begin in column 1 for FIND to locate the string. The format of the FIND command is:

FIND string

If no line beginning with string is found, the pointer stops at the end of the file, and the word BOTTOM is printed. The string cannot contain commas. For example:

```

FIND C
C This program generates the numbers 1 to 10
FIND 100
100 CONTINUE

```

The NFIND Command: The NFIND command moves the pointer to the first line below the current line which does not begin with string. The format of the NFIND command is:

NFIND string

For example:

```

PRINT 6
.NULL.
C This program generates the numbers 1 to 10
C and prints the numbers on the terminal screen.
C
      DIMENSION LNUMB (50)
      DO 100 I = 1, 10
TOP
NFIND C
      DIMENSION LNUMB (50)

```



Searching on a Specific Column: You can also find a string starting in a column other than column 1 by specifying the number of the column within parentheses directly after the command word.

FIND(n) string

The parentheses () around the column number are required. There cannot be any spaces between FIND and (n). For example:

```
PRINT 12
.NULL.
C This program generates the numbers 1 to 10
C and prints the numbers on the terminal screen.
C
  DIMENSION LNUMB (50)
  DO 100 I = 1, 10
    LNUMB (I) = I
    WRITE (1, 200) LNUMB (I)
100  FORMAT (10X, I5)
CONTINUE
CALL EXIT
END

TOP
FIND(7) D
  DIMENSION LNUMB (50)
FIND(2) 0
200  FORMAT (10X, I5)
```

Like FIND, you can use the NFIND command beginning on a column other than column 1 using the format:

NFIND(n) string

For example:

```
NFIND(1) C
  DIMENSION LNUMB (50)
FIND(2) 0
200  FORMAT (10X, I5)
NFIND(2) 0
  CALL EXIT
```

### Text-changing Commands

The APPEND, CHANGE, DELETE, INSERT, IB, RETYPE, OOPS, DUNLOAD, LOAD, and UNLOAD commands alter the text on one or several lines.



The APPEND Command: The APPEND command attaches a specified string to the end of the current line. The format of the APPEND command is:

APPEND string

Remember: One blank separates the command word APPEND (or abbreviation) from the string you wish to append. All further blanks are treated as part of the string. For example:

```
DIMENSION LNUMB (50)
APPEND , LSTORE (50)
DIMENSION LNUMB (50), LSTORE (50)
```

The CHANGE Command: The CHANGE command replaces one string in the current line with another string. The first character after the command word CHANGE (or abbreviation) is used as the delimiter. The format of the CHANGE command is:

CHANGE/string-1/string-2/[G] [n]

For example:

```
DIMENSION LNUMB (50), LSTORE (50)
CHANGE/DIMENSION/COMMON/
COMMON LNUMB (50), LSTORE (50)
```

Use a delimiter which does not occur in the text you are changing. Slash is a common delimiter, but if your text to be changed contains slashes, use a different character, as in this example:

```
DIMENSION LNUMB (50)/ LSTORE (50)
CHANGE;/;/
DIMENSION LNUMB (50), LSTORE (50)
```

If the letter G (for General) is specified, CHANGE will change every occurrence of string-1 on a line. If you do not specify G, only the first incidence of string-1 will be changed.

If the value of n is either 0 or 1, EDITOR only makes changes on the current line. (If n is either 0 or unspecified, the default value of 1 is used.) If a value other than 0 or 1 is specified, EDITOR will inspect and make changes on n lines starting at the current line, and leave the pointer positioned at the nth line. If there are fewer than n lines in the file, the message BOTTOM is printed. EDITOR prints out all changed lines, plus the last line examined.

#### Notes

1. Remember to issue the TOP command before making changes on the file as a whole.



2. If you end the command with a RETURN, you can omit the closing delimiter.
3. You can specify the semicolon (;) as a text character within the delimiters -- e.g., if you used "@" every place in your file where you wanted to use ";" then the command sequence TOP, CHANGE/@/;/G9999 would change all the @'s to ;'s. (Make sure that n is greater than the number of lines in your file.)
4. You can use CHANGE to insert characters at the beginning of a line with the sequence:

CHANGE//string/

For example:

```
LNUMB (50), LSTORE (50)
CHANGE//      DIMENSION /
      DIMENSION LNUMB (50), LSTORE (50)
```

The DELETE Command: The DELETE command deletes n lines, including the current line, and leaves the pointer at the null line where the last deleted line was. The null line is maintained, in case you wish to insert a new line, until a new command moves the pointer away. The format of the DELETE command is:

DELETE [n]

If n is not specified, the default value of 1 is used. n may be positive or negative, indicating deletion of the current line plus n-1 lines below or above the current line. Since n always indicates the current line, the commands D, D1, and D-1 are all equivalent. For example:

```
TOP
PRINT 5
.NULL.
C This program generates the numbers 1 to 10
C and prints the numbers on the terminal screen.
C
      DIMENSION LNUMB (50), LSTORE (50)
NEXT -2
C and prints the numbers on the terminal screen.
DELETE
PRINT
.NULL.
TOP
PRINT 4
.NULL.
C This program generates the numbers 1 to 10
C
      DIMENSION LNUMB (50), LSTORE (50)
```



The INSERT Command: The INSERT command inserts a specified new line following the current line; the inserted line then becomes the current line. The format of the INSERT command is:

INSERT newline

For example:

```

        DIMENSION LNUMB (50), LSTORE (50)
        DO 100 I = 1, 10
NEXT -1
        DIMENSION LNUMB (50), LSTORE (50)
INSERT      COMMON LSTART (50)
PRINT 2
        COMMON LSTART (50)
        DO 100 I = 1, 10

```

The IB Command: The IB command inserts a new line ahead of the current line; the inserted line then becomes the current line. The format of the IB command is:

IB newline

For example:

```

PRINT 5
.NULL.
C This program generates the numbers 1 to 10
C and prints the numbers on the terminal screen.
C
        DIMENSION LNUMB (50), LSTORE (50)
IB      COMMON LSTART (50)
NEXT -3
C This program generates the numbers 1 to 10
PRINT 5
C This program generates the numbers 1 to 10
C and prints the numbers on the terminal screen.
C
        COMMON LSTART (50)
        DIMENSION LNUMB (50), LSTORE (50)

```

The RETYPE Command: The RETYPE command deletes the current line and replaces it with the text specified in string. The format of the RETYPE command is:

RETYPE string



Remember: The first space after RETYPE separates the command word from the parameter; all further spaces are part of string. For example:

```
C Thsi porgarm gennratse hte numbers 1 too 1999
C and prints the numbers on the terminal screen.
C
NEXT -2
C Thsi porgarm gennratse hte numbers 1 too 1999
RETYPE C This program generates the numbers 1 to 10
PRINT
C This program generates the numbers 1 to 10
PRINT 3
C This program generates the numbers 1 to 10
C and prints the numbers on the terminal screen.
C
```

The string is terminated by either a semicolon (;) or a RETURN.

RETYPE followed immediately by a space and a RETURN erases the current line and replaces it with a blank line; RETYPE followed by a RETURN yields the reply: BAD RETYPE. For example:

```
C This program generates the numbers 1 to 10
C and prints the numbers on the terminal screen.
C
NEXT -1
C and prints the numbers on the terminal screen.
RETYPE
BAD RETYPE
RETYPE
PRINT

NEXT -1
C This program generates the numbers 1 to 10
PRINT 3
C This program generates the numbers 1 to 10

C
```

The OOPS Command: The OOPS command undoes the last line changed and reinstates it to its condition before the modification. This command does not work for changes to multiple lines at a time. The format of the OOPS command is:

OOPS



For example:

```

        DIMENSION LNUMB (50), LSTORE (50)
CHANGE/DIMENSION/COMMON
        COMMON LNUMB (50), LSTORE (50)
OOPS
        DIMENSION LNUMB (50), LSTORE (50)

```

The DUNLOAD Command: The DUNLOAD command creates a new file with the indicated filename, copies n lines from the EDITOR work file (beginning with the current line) into this new file, and then deletes these n lines from the work file. The format of the DUNLOAD command is:

DUNLOAD filename [n]

If filename is not specified, you will get the error message:

```

BAD FILE UNIT
BAD DUNLOA

```

#### Note

Be careful with DUNLOAD not to specify a filename currently in your directory unless you want the old file wiped out.

If n is not specified, the default value of 1 is used and one line is DUNLOADED. DUNLOAD leaves the pointer positioned at a null line where the deleted lines used to be; this null line disappears as soon as the pointer is moved.

The DUNLOAD command is useful for moving lines of text to different places; DUNLOAD can also be used instead of DELETE if you want to make sure that you do not accidentally delete large blocks of text. For example:

```

TOP
PRINT 12
.NULL.
        DIMENSION LNUMB (50), LSTORE (50)
        DO 100 I = 1, 10
            LNUMB (I) = I
            WRITE (1, 200) LNUMB (I)
C This program generates the numbers 1 to 10
C and prints the numbers on the terminal screen.
C
200   FORMAT (10X, I5)
100   CONTINUE
        CALL EXIT
        END
NEXT -6
C This program generates the numbers 1 to 10

```



DUNLOAD COMMENTS 3

TOP

PRINT 12

.NULL.

DIMENSION LNUMB (50), LSTORE (50)

DO 100 I = 1, 10

LNUMB (I) = I

WRITE (1, 200) LNUMB (I)

200 FORMAT (10X, I5)

100 CONTINUE

CALL EXIT

END

BOTTOM

The LOAD Command: The LOAD command copies the contents of a file (specified by its filename) into the EDITOR work file just below the current line. The pointer will then be just below the end of the loaded text, positioned at a null line. The format of the LOAD command is:

LOAD filename

LOAD does not affect the contents of the original file filename in any way; it simply copies the contents of filename into the work file. For example:

TOP

PRINT 3

.NULL.

DIMENSION LNUMB (50), LSTORE (50)

DO 100 I = 1, 10

TOP

LOAD COMMENTS

EDIT

TOP

PRINT 6

.NULL.

C This program generates the numbers 1 to 10

C and prints the numbers on the terminal screen.

C

DIMENSION LNUMB (50), LSTORE (50)

DO 100 I = 1, 10

#### Note

Loaded text will not go in your permanent files in your UFD unless you FILE at the end of the editing session. (The FILE command is discussed in detail later in this chapter.)



The UNLOAD Command: The UNLOAD command copies n lines beginning at the current line from the file being EDITED into a new file named filename. The format of the UNLOAD command is:

UNLOAD filename [n]

If n is 0 or omitted, it is assumed to be 1. A negative value for n unloads the preceding n-1 lines and the current line, in the correct order.

The last line unloaded is the new current line. Be careful not to specify a filename currently in use unless you want the old file wiped out.

UNLOAD does not delete the lines of the work file as it writes these lines into the file filename. For example:

```

TOP
PRINT 6
.NULL.
C This program generates the numbers 1 to 10
C and prints the numbers on the terminal screen.
C
      DIMENSION LNUMB (50), LSTORE (50)
      DO 100 I = 1, 10
NEXT -4
C This program generates the numbers 1 to 10
UNLOAD TEMP 3
PRINT 2
C
      DIMENSION LNUMB (50), LSTORE (50)
TOP
PRINT 6
.NULL.
C This program generates the numbers 1 to 10
C and prints the numbers on the terminal screen.
C
      DIMENSION LNUMB (50), LSTORE (50)
      DO 100 I = 1, 10

```

### Ending and Saving an EDITOR Session

The QUIT, FILE, and SAVE commands end and/or save the current EDITOR session.

The QUIT Command: The QUIT command tells the EDITOR you do not want to save the EDITOR work file; instead you want to preserve the original and to return to PRIMOS-level. The format of the QUIT command is:

QUIT



If you have created or modified a file during the session, EDITOR responds to a QUIT with:

FILE MODIFIED, OK TO QUIT?

This message asks whether EDITOR may throw away the work file.

A YES (or Y, YE, O, OK, or NULL line RETURN) response returns you to PRIMOS without saving the current session's editing. Any other response provokes a PLEASE FILE message. (See the explanation of the FILE command.) If you did not create or modify a file, saying QUIT automatically returns you to PRIMOS.

The FILE Command: The FILE command turns the EDITOR work file (which is so far only a temporary file) into a permanent file in your UFD and returns you to PRIMOS.

#### WARNING

Since the work file does not exist outside of EDITOR, you must FILE if you want to save your work. If you do not FILE or SAVE your work, it will be destroyed.

The format for the FILE command is:

FILE [filename]

If you have been creating a new file, you must specify filename. (The error message FILENAME MUST BE SPECIFIED occurs if you do not.)

You cannot have two files with the same name in the same UFD. If you give a filename which already exists in your UFD, EDITOR will delete the old file of that name from your UFD (without any warning), and put the EDITOR work file in its place.

The same warning holds true for old files. If you have been working on an old file, and you specify the old filename, or say FILE without any filename, your old copy will be deleted, and your new version kept. Giving a new filename keeps both the old and new versions.

The rules for making filenames are:

- Filenames can be up to 32 characters long.
- Filenames can contain only the following characters:  
     A through Z  
     0 through 9  
     & - \$ \* . \_ / #
- The first character may be any legal character except a digit.



- Characters NOT permitted in filenames include: imbedded blanks and special characters such as ? ! @ ; , "
- Uppercase and lowercase letters are treated as uppercase by PRIMOS. (Letters entered in lowercase are converted to uppercase.)

Valid Filenames

NEWFILE  
 Todays-Prices  
 Highs&Lows  
 \$monthly.REPORT  
 R34587  
 A-tale-of-two-cities

Invalid Filenames

A?  
 Two@John  
 "Eureka"  
 Why a Duck  
 8ball

The FILE command can also be used to make copies of any file, simply by typing ED plus the filename and filing the copied work file immediately with a new filename, as in:

```
OK, ED FIN.TEST
EDIT
FILE TEST
```

OK,

The SAVE Command: The SAVE command writes the contents of the current EDITOR session into a filename but does not leave the EDITOR or terminate the current session. The format of the SAVE command is:

SAVE filename

filename is the name of the file you want the current session copied to. If filename is not specified, the current EDITOR session is written into the filename being edited. The SAVE command is useful if you are editing a file extensively and want to ensure that your work is saved periodically as you go along.

SAMPLE EDITING SESSIONS

Here are three examples showing the writing and editing of source files.



A PL/I Example

OK, ED  
INPUT

EDIT  
TABSET 3 6 9 12 18 21  
MODE NOSEMI

INPUT  
\DO I=1 TO 10  
\\DO J=1 TO 10;  
\\\X(I,J)=A(I)+B(I);  
\\\Y(I,J)=SQRT(X(I,J));  
\\END; /\*J-LOOP\*/  
\\END; /\*I-LOOP\*/

EDIT  
TOP  
NEXT  
DO I=1 TO 10  
APPEND ;  
DO I=1 TO 10;  
NEXT 2  
X(I,J)=A(I)+B(I);  
CHANGE/J/J)  
X(I,J)=A(I)+B(I);

TOP  
PRIT ?PRINT 99  
.NULL.  
DO I=1 TO 10;  
DO J=1 TO 10;  
X(I,J)=A(I)+B(I);  
Y(I,J)=SQRT(X(I,J));  
END; /\*J-LOOP\*/  
END; /\*I-LOOP\*/

BOTTOM  
FILE ED.EX

OK,

An empty line puts us in EDIT mode.  
Set tabs for PL/I code.  
Changes semicolon character from EDITOR delimiter to literal character.  
Empty line puts us in INPUT mode.  
Type in source code using tabs to show levels of indentation.

Go to top of file.  
First non-null line.

Add a forgotten semicolon.

Down two lines.

Balance parentheses.

Check code before filing.

Name a new file when you file it.



A FORTRAN Example

OK, ED  
INPUT

EDIT  
TABSET 7 45

INPUT  
\A-"=30\/\* COMMENT

\B=40  
C-A?\C=A+B

\PRINT 10,C  
CALL EXTI""IT  
\END

EDIT  
FILE FIN.TEST  
OK, ED FIN.TEST  
EDIT  
PRINT 20  
.NULL.

A=30  
B=40  
C=A+B  
PRINT 10,C

CALL EXIT  
END

BOTTOM  
NEXT -3  
PRINT 10,C

TABSET 7 45  
INSERT 10\FORMAT('THE ANSWER IS',I4)  
TOP, PRINT 20  
.NULL.

A=30  
B=40  
C=A+B  
PRINT 10,C

10 FORMAT('THE ANSWER IS',I4)  
CALL EXIT  
END

BOTTOM  
FILE

FIN.TEST

OK,

Useful settings for FORTRAN.

Quote mark erases one character.

Question mark erases entire line.

/\* COMMENT

Move up three lines.

Set FORTRAN tabs again.  
Insert forgotten line.  
Check file once more.

/\* COMMENT

No need to use a filename this time.



A COBOL Example

OK, ED  
INPUT

EDIT  
MODE COLUMN

INPUT

	1	2	3	4	5	6	7
12345678901234567890123456789012345678901234567890123456789							

ID DIVISION.

PROGRAM-ID. TEST.

INSTALLATION. PRIME.

Source coding is keyed in,  
aligned by column.

\ \*

.  
.  
.

The first tab default is position  
6. A space after the backslash  
character positions the asterisk  
in the continuation column 7.

EDIT  
FILE COBOL.TEST

OK, ED COBOL.TEST  
EDIT  
PRINT 23  
.NULL.

ID DIVISION.

PROGRAM-ID. TEST.

INSTALLATION. PRIME.

\*

BOTTOM  
FILE  
COBOL.TEST

OK,

LISTING PROGRAMSTerminal Listing

Source programs may be listed at the terminal by using the SLIST command, described in Chapter 3.

Line Printer Listing

Use the SPOOL command (explained below) to obtain a hard copy of a source file on the system line printer.



Renaming

Programs may be renamed with the PRIMOS command CNAME (Chapter 3). You must have delete (D) and add (A) access in a directory containing the programs in order to use this command.

Deleting

Programs may be deleted with the PRIMOS command DELETE (Chapter 3). You must have delete access in order to use this command.

PRINTING HARD COPY WITH THE SPOOL COMMAND

Printed copies of files from a line printer (also called hard copies) are obtained with the SPOOL command. It has several options, some of which will not apply to all systems, as systems may be configured differently. The format is:

SPOOL pathname [options]

PRIMOS makes a copy of pathname in the Spool Queue for the line printer, and displays the message:

PRTnnn spooled, records: x, name: filename

nnn is a three-digit number which identifies the file in the Spool Queue. x is number of records in the file. PRIMOS spools out short files as soon as possible; long files receive a lower priority. For example:

OK, SPOOL SALESREPORT

[SPOOL rev 19.0]

PRT015 spooled, records: 87, name: SALESREPORT

OK, SPOOL BEECH>BRANCH6>TWIG3

[SPOOL rev 19.0]

PRT016 spooled, records: 1, name: TWIG3

OK,

In this example, one file was spooled by filename and the other by pathname. However, SPOOL will refer to both by their filenames, that is, EXAMPLE and TWIG3. If these files are submitted in the order above, TWIG3 will probably receive the higher priority (as the shorter file) and will be printed first. (Exact priority settings are determined for each system by the System Administrator.)



Checking the Queue

To check the status of the Spool Queue, give the command:

SPOOL -LIST

PRIMOS returns a list of all the files in the Queue which have not yet been printed. Additional information, such as the size, destination, the PRT number, any options, the form-type and the user-id of the user who spooled the file, are also specified. For example:

OK, SPOOL -LIST  
[SPOOL rev 19.0]

System	SYSX									
user	prt	time	name	size	opts/#	form	defer	at:	MAIN	
DOUG	001	13:31	REPORT	210	3		22:00			
KEITH	002	14:16	SKETCH	1		PLOT				
PETE	003	14:17	PROGA.LIST	8					OFFICE	
JANE	004	14:17	TIME_TABLE	40	5	WIDE			FLOOR2	
WRITER	005	14:18	CHAPTER3	57						
MONEY	006	14:19	BUDGET	15		WHITE	18:00			
DORIS	007	14:19	SORT.COBO	12					FLOOR3	
ORDERS	008	14:22	CONSOLIDATED	10	2					

OK,

Cancelling a Spool Request

To cancel one or more spool requests, the command format is:

SPOOL -CANCEL [PRT]n1 [n2...]

n1, n2, and so on, are the numbers of your spool files to be cancelled.  
For example:

OK, SPOOL -CANCEL 47 048 PRT049  
[SPOOL rev 19.0]  
PRT047 cancelled.  
PRT048 cancelled.  
PRT049 cancelled.

Printing Multiple Copies

You can request several copies of one file by using the -COPIES option:

SPOOL pathname -COPIES n

n is the number of copies desired.



Deferring Printing

The -DEFER option tells the Spooler not to begin printing the indicated file until the time specified with DEFER. This permits you to enter SPOOL requests at your convenience, rather than waiting for the appropriate hour.

Specify the DEFER option by:

SPOOL pathname -DEFER time

The format for time is hh [:] mm [AM/PM]. If AM or PM is given, hh:mm (the colon is optional) must be in 12-hour format (for example, 1000 PM). Otherwise, time will be interpreted as 24-hour format (in which 2200 is 10:00 PM and 1000 is 10:00 AM).

Printing on Special Forms

Line printers traditionally use one of two types of paper — "wide" listing paper, on which most program listings appear, and 8-1/2 x 11-inch white paper, which is standard for memos and documentation. Computer rooms often stock a variety of special paper forms for special purposes, such as five-copy sets, preprinted forms (checks, orders, invoices), or odd sizes or colors of paper.

Request a specific form by:

SPOOL pathname -FORM form-name

form-name is a six-character (or fewer) combination of letters that defines a specific form of paper for your system. A list of form names available on your system can be obtained with the PROP command, which is explained in the PRIMOS Commands Reference Guide.

Changing the Header

The -AS option tells the Spooler to print your file under a different name. The form is:

SPOOL pathname -AS alias

The alias will appear on the header and in the SPOOL -LIST display.



Printing at Specific Locations

Networks with several printers often arrange to have the printers read each other's queues. It is therefore possible for a spool request to be printed at another location, perhaps many miles distant. To insure that a spool request is printed where you want it, use the -AT option:

SPOOL pathname -AT destination

destination is a word of 16 letters or fewer. A list of available destination names can be obtained with the PROP command, explained in the PRIMOS Commands Reference Guide. If a destination appears in the heading of the SPOOL -LIST display (for example, at: MAINROOM), then that destination is the default destination for spool requests. If no destination follows "at:", then no default has been established; spool requests without destinations may be intercepted by any available printer.

Eliminating Headers

To have files printed without header or trailer pages, use the -NOHEAD option:

SPOOL pathname -NOHEAD

This option is particularly useful with preprinted forms, but if you are using this option in a multiuser environment, you will have to identify your own jobs.

Multiple Options

Any or all of the above options may be used jointly in a single SPOOL command line. If -LIST or -CANCEL is included, it must be the last option on the command line. For example:

```
OK, SPOOL TEST -AS EX.1 -AT BLDG.1 -DEFER 22:00
[SPOOL rev 19.0]
PRT048 spooled, records: 1, name: TEST
```

This particular command requests that the file named "TEST" be printed at the "BLDG.1" printer, under the alias of "EX.1", at 10 PM (22:00).



Section 1: Introduction

The purpose of this document is to provide a comprehensive overview of the project's objectives, scope, and timeline. This document is intended for use by all project team members and stakeholders.

Section 2: Project Objectives

The primary objective of this project is to develop a new software application that will streamline the workflow of the department. The project will also aim to improve communication and collaboration among team members.

Section 3: Scope of Work

The scope of work for this project includes the design, development, testing, and deployment of the software application. It also includes the training of end users and the documentation of the system.

Section 4: Timeline

The project timeline is as follows: Design - 2 weeks, Development - 6 weeks, Testing - 2 weeks, Deployment - 1 week. The total project duration is 11 weeks.

Section 5: Resources

The resources required for this project include a dedicated project manager, a software developer, a tester, and a system administrator. Additionally, hardware and software licenses are also required.

Section 6: Risk Management

The risks associated with this project are low. The main risk is the potential for delays in the development phase, which can be mitigated by regular communication and progress reporting.

This document is a confidential document and should be handled accordingly. It contains information that is not to be distributed outside the project team.



# 5

## Compiling Programs

### INTRODUCTION

After the source code has been entered into the system, it must be compiled. Compilation creates a new file of linkable code--the object (or binary) file. Each high-level language has its own compiler which creates object code from source code. At the object code level, these languages are equivalent. Thus, modules of object code originating from different source languages may be linked together to form a run-time program. (Further comments on this will be found at the end of this chapter.) Details of Prime's compilers are treated in the individual language guides. This chapter discusses features common to all compilers.

Prime's convention for source file names uses compiler-name suffixes. The suffixes are separated from the "base name" of the file by a dot. Thus, a FORTRAN IV program with a base name DRAGON would be named DRAGON.FIN, while a FORTRAN 77 program with a base name WYVERN would be named WYVERN.F77. This naming convention helps you keep track of the types of files in your directories. It also helps you access groups of files using wildcards. (Wildcards are explained in Chapter 18.)

Table 5-1 lists the compiler and loader suffixes recognized by Prime software.



Table 5-1  
Recognized Language  
and Compiler/Loader Filename Suffixes

Suffix	Meaning	Recognized by:	Supplied by:
BASIC	BASIC/VM source file	BASIC/VM	user
COBOL	COBOL source file	COBOL compiler	user
F77	FORTRAN 77 source file	F77 compiler	user
FTN	FORTRAN IV source file	FTN compiler	user
PASCAL	Pascal source file	PASCAL compiler	user
PL1G	PL/I, Subset G source file	PL1G compiler	user
PMA	PMA source file	PMA assembler	user
RPG	RPG II source file	RPG compiler	user
RPG	RPG II V-mode source file	VRPG compiler	user
BIN	Binary file (created by compiler)	LOAD, SEG	compilers or user
LIST	Listing file (created by compiler)		compilers, SPSS, user
SAVE	Runfile created by R-mode loader, LOAD	LOAD, RESUME	LOAD or user
SEG	Segment directory created by SEG	SEG	SEG or user



INVOKING THE COMPILER

The compiler is invoked from PRIMOS command level by the command:

compiler pathname [options]

compiler is the compiler for the language in which the source program is written. Current compilers are:

<u>Compiler</u>	<u>Language</u>
COBOL	COBOL
F77	FORTRAN 77
FTN	FORTRAN IV
Pascal	Pascal
PLlG	PL/I Subset G
RPG	RPG II
VRPG	RPG II (V-mode Compiler)

pathname is the pathname of the source program file. Each compiler recognizes its own suffix. Therefore, you do not have to specify the suffix when you are invoking the compiler named by the suffix. When a compiler is invoked, it looks first for pathname plus its suffix. If the filename with a suffix is not found, the compiler then looks for pathname without the identifying suffix. For example, typing "FTN DRAGON" causes the FTN compiler to look first for DRAGON.FTN. If it doesn't find DRAGON.FTN, it then looks for DRAGON.

options allow you to specify the following:

- the creation of object and listing files,
- the mode in which the object code is to be generated,
- the types of cross references and listings to be generated,
- debugger interfaces.

These options may be common to all compilers or unique to a particular language. The common options are summarized in Table 5-2 and discussed in the following paragraphs.



Table 5-2  
Compiler Defaults

Compiler	Binary/ Object File	Listing File	Cross- Reference	Mode
COBOL	yes	yes	no	64V
F77	yes	no	no	64V
FTN	yes	no	no	32R
Pascal	yes	no	no	64V
PLLG	yes	no	no	64V
RPG	yes	yes	yes	64R
VRPG	yes	no	no	64V

### OBJECT FILES

In all compilers, the default is to create an object file. The default name of an object file depends on the name of the source file. If the name contains a compiler-identifying suffix, the object file name is filename.BIN. (For example, the object filename of PAYROLL.COBOL is PAYROLL.BIN.) If the name contains no suffix, the default object filename is B\_filename.

A non-default binary filename can be created (or suppressed) with the -BINARY option (abbreviation -B). This allows you to use the .BIN suffix even if the source filename is not suffixed. Possible arguments for this option are:

<u>Argument</u>	<u>Meaning</u>
-BINARY YES	Creates a binary file with default name.
-BINARY NO	Does not create a binary file.
-BINARY pathname	Creates a binary file called <u>pathname</u> .



LISTING FILES

Each compiler can create a file listing the source program. Language-specific options are available to expand on these listings and add more information. The option to create a listing file is `-LISTING` (abbreviation `-L`). The default name, which is formed in the same way as the default object file name, is `filename.LIST` (or `L_filename`, if the source filename has no suffix). The arguments for the `-LISTING` option are:

<u>Argument</u>	<u>Meaning</u>
<code>-LISTING YES</code>	Creates a listing file with default name.
<code>-LISTING NO</code>	Does not create a listing file.
<code>-LISTING pathname</code>	Creates a listing file called <u>pathname</u> .
<code>-LISTING TTY</code>	Prints the listing file at your terminal.
<code>-LISTING SPOOL</code>	Prints the listing file on a line printer.

CROSS REFERENCES

Each language has its particular cross-reference listing. Each lists the program's variables, shows where they appear in the program, and provides other useful information. Specific details are in each language guide. Cross references are listed by default for RPG only. In other languages, the cross-references listing is generated by using the option `-XREF` in the command line. The cross-reference listing is appended to the program listing file.

CODE GENERATION

The addressing mode in which object code is to be loaded is chosen at compilation time. R-mode is the addressing mode used for a single segment program. A single segment program cannot exceed 128K bytes. R-mode is used mainly to create external commands.

V-mode is the addressing mode used for a multi-segmented program, which can be as large as 32MB. Programs in V-mode can take full advantage of the virtual address space.

Table 5-3 shows which types of code can be generated by each compiler.



Table 5-3  
Code Generation

	32I	64V	64R	32R
FORTRAN 77 (F77)	X	X		
FORTRAN IV (FTN)		X	X	X
Pascal	X	X		
PL/I Subset G	X	X		
COBOL		X		
RPG			X	
VRPG		X		

In general, 64V mode is the mode of choice. This is the default on all compilers except FORTRAN IV (FTN) and RPG II (RPG). At present, the RPG compiler generates only 64R mode code. To generate 64V mode code in FORTRAN IV, use the 64V option in the command line. For example:

```
FTN GOOD -LISTING YES -64V
```

compiles the program GOOD.FTN, producing 64V mode code and creating a listing file, GOOD.LIST.

The FORTRAN 77 (F77), Pascal, and PL/I (PLIG) compilers can also generate 32I mode code. 32I mode code handles double-precision floating-point arithmetic more rapidly than the other modes do. Therefore, it is the mode of choice for many mathematical calculations. To generate 32I mode code, use the 32I option in the command line, as in:

```
F77 CHEERS -32I
```

#### LOADING

All code generated in 64V or 32I mode is loaded with SEG. (This procedure is often called linking on other systems.) Code generated in 32R or 64R mode is loaded with LOAD. These loaders (or linkers) are summarized in Chapter 6, and explained in detail in the LOAD and SEG Reference Guide.



COMPILER MESSAGES

If an interactive compilation completes successfully, a message to that effect is printed at the user's terminal. (If the compilation is not interactive, the message may be printed into the user's COMOUTPUT file. See Chapter 10 for information on COMOUTPUT files.) If compilation is not successful, error and/or warning messages will indicate the offending line and the type of error. Some severe errors halt the compilation as soon as they are discovered. Others allow the compilation to proceed. Each compiler has its own error messages.

Error messages printed by the F77 and PL1G compilers include explanatory comments. Error messages generated by the FTN, COBOL, Pascal, and RPG compilers are discussed in those language guides.

COMBINING LANGUAGES IN A PROGRAM

Since all high-level languages are alike at the object code level, and since all use the same calling conventions, programs compiled by the Pascal, FTN, F77, COBOL, or PL1G compilers can call subroutines compiled by any of the other three compilers. For example, a program written in COBOL could call a subroutine written in FORTRAN 77 which might use a utility subroutine written in PL/I, Subset G. Procedures compiled by the high-level language compilers may also call, or be called by, procedures written in Prime's assembler language, PMA. The following cautions, however, should be observed:

- All I/O routines should be written in a single language.
- Be sure that there is no conflict in data types for variables being passed as arguments. For example, an integer in FORTRAN should be declared as fixed binary in PL/I. Also, remember that PL/I and COBOL may not interpret structures identically.
- All procedures within a program must use compatible addressing modes. Do not put R-mode procedures into a V-mode or I-mode program, or vice versa. However, V-mode and I-mode are compatible within programs.
- Some special restrictions must be observed when FTN and F77 routines are linked together. These are discussed in the FORTRAN 77 Reference Guide.

A complete discussion of parsing data types between languages appears in the Subroutines Reference Guide.



General Notes

The following notes are for the purpose of providing information to the reader regarding the data presented in the tables. The data were obtained from the records of the Department of the Interior, Bureau of Land Management, and are presented in the tables in the order in which they were received. The data are presented in the tables in the order in which they were received. The data are presented in the tables in the order in which they were received.

The data are presented in the tables in the order in which they were received. The data are presented in the tables in the order in which they were received. The data are presented in the tables in the order in which they were received.

Notes on the Data

The data are presented in the tables in the order in which they were received. The data are presented in the tables in the order in which they were received. The data are presented in the tables in the order in which they were received.

The data are presented in the tables in the order in which they were received. The data are presented in the tables in the order in which they were received. The data are presented in the tables in the order in which they were received.

The data are presented in the tables in the order in which they were received. The data are presented in the tables in the order in which they were received. The data are presented in the tables in the order in which they were received.

The data are presented in the tables in the order in which they were received. The data are presented in the tables in the order in which they were received. The data are presented in the tables in the order in which they were received.

The data are presented in the tables in the order in which they were received. The data are presented in the tables in the order in which they were received. The data are presented in the tables in the order in which they were received.

The data are presented in the tables in the order in which they were received. The data are presented in the tables in the order in which they were received. The data are presented in the tables in the order in which they were received.



# 6

## Loading Programs

### INTRODUCTION

PRIMOS has two utilities for loading programs: SEG and LOAD. SEG loads and runs V-mode and I-mode programs; LOAD loads R-mode programs. This chapter explains the basic use of SEG and LOAD for programs written in high-level languages. Language-specific aspects of loading programs are treated in the individual language guides. The loaders are explained in detail in the LOAD and SEG Reference Guide.

### SEG

The PRIMOS SEG utility converts object modules (such as those generated by the FTN, F77, COBOL, Pascal, and PL1G compilers) into segmented runfiles that execute in the 64V addressing mode and take full advantage of the architecture and instruction set of the Prime 50 series. Segmented runfiles offer the following four advantages:

- Much larger programs: up to 256 segments per user program (32 megabytes) can be written.
- Permits access to V-mode instructions and architecture for faster execution.



- Ability to install shared code: a single copy of a procedure can service many users, significantly reducing paging time.
- Re-entrant procedures permitted: procedure and data segments can be kept separate.

The following description emphasizes the commands and functions that are of most use to high-level language programmers. Extended features, as well as a complete description of all SEG commands, including those for advanced system-level programming, are described in the LOAD and SEG Reference Guide.

#### USING SEG UNDER PRIMOS

SEG is invoked by PRIMOS command:

SEG [pathname]

or

SEG -LOAD

A pathname is given only when an existing SEG runfile is to be executed. (See Chapter 7.) Otherwise, the command transfers control to SEG command level, which prints a # prompt character and awaits a SEG subcommand. After executing a subcommand successfully, the loader repeats the prompt character. (SEG's loader prints a \$ prompt to request its subcommands.)

When the -LOAD option is used, SEG enters the loader automatically and requests LOAD subcommands. Using this option also causes SEG to create a segment directory with the filename x.SEG, where x is the base name of the first file loaded. (Details and examples of this loading process are shown later in this chapter.)

If an error occurs during an operation, SEG prints an error message, then the prompt character. Error messages and suggested handling techniques are discussed in this chapter and in Appendix D.

When a system error (File in use, Illegal name, Insufficient access rights, etc.) is encountered, SEG prints the system error and returns the prompt symbol.

SEG remains in control until a QUIT subcommand returns control to PRIMOS, or an EXECUTE subcommand starts execution of the loaded program.

You can use SEG subcommands and comment lines in command files.



NORMAL LOADING

Loading is normally a simple operation with only a few straightforward commands needed. (SEG has many additional features to optimize runfile size or speed, perform difficult loads, load for shared procedures, and deal with possible complications. These are described in the LOAD and SEG Reference Guide.)

The following commands accomplish most loading functions.

SEG-level Commands

DELETE	Deletes segmented runfile.
HELP	Prints a list of SEG commands at terminal.
LOAD	Invokes loader subprocessor for entry of subcommands.

LOAD Subcommands

LOAD pathname	Loads specified object file.
LIBRARY [filename]	Loads library object files from UFD LIB. (Default is PFTNLB and IFTNLB.)
MAP [option]	Prints loadmap. Option 3 shows unresolved references (usually subroutines which have not been loaded). Mapping is explained in the <u>LOAD and SEG Reference Guide</u> .
INITIALIZE	Returns loader to starting condition in case of command errors or faulty load.
SAVE	Saves loaded memory image as runfile.
RETURN	Returns to SEG command level.
QUIT	Return to PRIMOS.



Note

SEG recognizes the .BIN suffix for binary files. Thus, to tell SEG to load the file DRAGON.BIN, you need only type "LO DRAGON". SEG also recognizes the .SEG suffix for subcommands that take segment directories as input files: for example, DELETE and RESTORE. Thus, saying:

```
OK, SEG
# DELETE DRAGON
```

deletes DRAGON.SEG, if a segment directory of that name exists. If not, SEG looks for the segment directory DRAGON, and deletes that segment directory if found.

Most loads can be accomplished by the following basic procedure:

1. Give the command SEG -LOAD.
2. Use the LOAD subcommand to load the object file and any separately compiled subroutines.
4. Use the LIBRARY subcommand to load subroutines called from libraries.
5. If you do not receive a LOAD COMPLETE message, do a MAP 3 to identify the unsatisfied references, and load them. If the unsatisfied references are the result of having misspelled some subroutine names in loading, you may want to initialize and repeat the loading process. If you have misspelled subroutines in your program, you will need to edit, recompile, and then reload.
6. When you have received the LOAD COMPLETE message, SAVE the runfile. SEG will give the runfile the default name filename.SEG, where filename is the name (without suffix) of the first object file loaded.

An example of such a load is:

```
OK, SEG -LOAD
$ LO DRAGON
$ LI
LOAD COMPLETE
$ SA
$ QU
OK,
```

If you want to specify a name other than the default name for the segment directory, use the following sequence for loading:

1. Invoke SEG from PRIMOS level.



2. Enter the LOAD command to initiate the loading process. At this point, SEG requests a name for the segment directory to be created. Type in the name you desire. (SEG will add the .SEG suffix automatically.)
3. Use the LOAD subcommand to load the object file and any separately compiled subroutines.
4. Use the LIBRARY subcommand to load subroutines called from libraries.
5. If you do not receive a LOAD COMPLETE message, do a MAP 3 to identify the unsatisfied references, and load them. If the unsatisfied references are the result of having misspelled some subroutine names in loading, you may want to initialize and repeat the loading process. If you have misspelled subroutines in your program, you will need to edit, recompile, and then reload.
6. SAVE the runfile.

If these commands produce a LOAD COMPLETE message, then loading was accomplished. If there is a problem, it will become apparent by the absence of a LOAD COMPLETE message or some other SEG error message. (See Appendix D for a complete list of all SEG error messages and their probable cause and correction.)

An example of this loading sequence is:

```
OK, SEG
[SEG REV 18.1]
# LOAD
SAVE FILE TREE NAME: #BENCH9
$ LO B_BENCH9
$ LI
LOAD COMPLETE
$ SA
$ OU
OK,
```

After a successful load, you can either start runfile execution from loader command level, or quit from the loader and start execution at PRIMOS command level by giving the command SEG pathname.

### Order of Loading

The following loading order is recommended:

1. Main program
2. Separately compiled user-generated subroutines (preferably in order of frequency of use)



3. Language-specific libraries (PL1GLB for PL/I, PASLIB for Pascal, VCOBLB and possibly NCOBLB for COBOL)
4. Other Prime Libraries (LI filename), such as VAPPLB(V-mode applications library), VSRTLI (V-mode sort library), VKDALB (MIDAS library)
5. Standard Prime library (LI)

For example, a COBOL program which uses MIDAS files would be loaded as follows:

```
OK, SEG -LOAD
[SEG rev 19.0]
$ LOAD MAIN      Main program is loaded first.
$ LOAD SUBR      Separately compiled subroutine is loaded next.
$ LI VCOBLB      Shared COBOL library is always used.
$ LI NCOBLB      Non-Shared library is used with separately-compiled
                   subroutines.
$ LI VKDALB      MIDAS library is used with MIDAS files.
$ LI             Standard (FORTRAN) library is used with any language.
LOAD COMPLETE
$ SAVE           Save the file image
$ QUIT          Return to PRIMOS command level.
OK,
```

#### THE R-MODE LOADER

The PRIMOS LOAD utility converts object modules, such as those generated by the FIN or RPG compilers, into runfiles that execute in the 32R or 64R addressing modes. (Runfiles to execute in the 64V mode must be loaded using the segmentation utility, SEG.)

LOAD recognizes the .BIN suffix for object files. For example, if you give the name FROG, LOAD looks first for FROG.BIN. If it does not find FROG.BIN, it looks for FROG.

LOAD uses the .SAVE suffix for the runfiles it creates.

The following description emphasizes the loader commands and functions that are of most use to the FORTRAN and RPGII programmer. For a complete description of all loader commands, including those for advanced system-level programming, refer to the LOAD and SEG Reference Guide.



USING THE LOADER UNDER PRIMOS

The PRIMOS command:

LOAD

transfers control to the R-mode loader, which prints a \$ prompt character and awaits a loader subcommand. After executing a command successfully, the loader repeats the \$ prompt character.

If an error occurs during an operation, the loader prints an error message, then the \$ prompt character. Loader error messages and suggested handling techniques are discussed in Appendix D. Most of the errors encountered are caused by large programs where the user is not making full use of the loader capabilities.

When a system error (File in use, Illegal name, Insufficient access rights, etc.) is encountered, the loader prints an error message and then the \$ prompt character.

The loader remains in control until a QUIT or PAUSE subcommand returns control to PRIMOS, or an EXECUTE subcommand starts execution of the loaded program.

Load subcommands can be used in command files, but comment lines result in a CM (command error) message unless they are preceded by an asterisk and a blank space (\* ).

NORMAL LOADING

Loading is normally a simple operation with only a few straightforward commands needed. The loader also has many additional features to optimize runfile size or speed, perform difficult loads, and deal with possible complications. For details on these, see the LOAD and SEG Reference Guide.

The following commands accomplish most loading functions.

PRIMOS-level Commands

FILMEM	Initializes user space in preparation for load.
LOAD	Invokes loader for entry of subcommands.
RESUME	Starts execution of a loaded, saved runfile.



LOAD Subcommands

DC	Defers loading of COMMON until everything else has been loaded. This prevents overlap of COMMON and program areas.
MODE option	Sets runfile addressing mode as D32R (default) or D64R.
LOAD pathname	Loads specified object file.
LIBRARY [filename]	Loads library object files from UFD LIB. (Default is FINLIB.)
MAP [option]	Prints loadmap. Option 3 shows unresolved references.
INITIALIZE	Returns loader to starting condition in case of command errors or faulty load.
SAVE [pathname]	Saves loaded memory image as runfile. If <u>pathname</u> is not given, LOAD creates a filename from the name (without suffix) of the first object file loaded plus the .SAVE suffix.
QUIT	Return to PRIMOS.

Most loads can be accomplished by the following basic procedure:

1. Use the PRIMOS FILMEM command to initialize memory.
2. Invoke LOAD.
3. Use the MODE command to set the addressing mode, if necessary. (The default is 32R mode.)
4. Use loader's LOAD subcommand to load the object file and any separately compiled subroutines. (LOAD will search first for filenames plus the .BIN suffix, then for the given filenames without the suffix.)
5. Use loader's LIBRARY subcommand to load subroutines called from libraries. The default library is FINLIB in the UFD LIB. Other libraries, such as SRTLIB or APPLIB, must be named explicitly.
6. If you do not have a LOAD COMPLETE, do a MAP 3 to identify the unsatisfied references, and load them. (If the DC option is used, the LOAD COMPLETE message may not appear until the SAVE command has been given.)



7. SAVE the runfile, either by giving an appropriate name, or by allowing LOAD to create a default filename. (The default is the name, without suffix, of the first object file loaded, plus the .SAVE suffix.)

If these commands produce a LOAD COMPLETE message, then loading was accomplished. If there is a problem, it will become apparent by the absence of a LOAD COMPLETE message or some other loader error message. (See Appendix D for a complete list of all loader error messages and their probable cause and correction.)

After a successful load, you can either start runfile execution from LOAD command level, or quit from the loader and start execution through the PRIMOS RESUME command. An example of such a load is:

```
OK, FILMEM
OK, LOAD
$ DC
$ LO WYVERN
$ LI
LOAD COMPLETE
$ SA
$ QU
OK,
```

### Order of Loading

The order of loading and procedures for mapping are the same for LOAD as they are for SEG.



The first of these is the fact that the  
the first of these is the fact that the  
the first of these is the fact that the

the first of these is the fact that the  
the first of these is the fact that the  
the first of these is the fact that the

the first of these is the fact that the  
the first of these is the fact that the  
the first of these is the fact that the

1942-1943

1942-1943

1942-1943

1942-1943

1942-1943

the first of these is the fact that the  
the first of these is the fact that the  
the first of these is the fact that the



# 7

## Running Programs Interactively

### INTRODUCTION

This chapter discusses the following topics:

- Choice of program environments
- Use of the SEG command to execute segmented runfiles
- Use of the RESUME command to run R-mode runfiles
- Run-time error messages

### PROGRAM ENVIRONMENTS

Under PRIMOS, programs may execute in three environments:

- Interactive
- Phantom
- Batch



### Interactive

This is the environment most frequently used. Program execution is initiated directly by the user, and the terminal is dedicated to the program during execution. The program accepts input from the terminal and prints output or error messages at the terminal. Major uses are:

- Program development and debugging
- Programs requiring short execution time
- Data entry programs, such as order entry or payroll
- Interactive programs, such as the Editor

### Phantom User

The phantom environment allows programs to be executed while "disconnected" from the terminal. Jobs run as phantoms accept input from a command file instead of the terminal. Output directed to the terminal is either ignored or directed to a file.

Major uses of phantoms are:

- Programs requiring long execution time, such as sorts
- Certain system utilities, such as line printer spooler
- Freeing terminals for other uses

### Batch Jobs

Since the number of phantoms on a system is limited, phantoms are not always available. The Batch environment, discussed in Chapter 11, allows users to submit non-interactive command files as Batch jobs at any time. The Batch monitor (itself a phantom) queues these jobs and runs them as phantoms become free.

### EXECUTING PROGRAMS

The existence of the interactive, phantom, and Batch environments allow you to choose how your programs are executed.



### Interactive Execution

Users can execute programs directly by using the RESUME or SEG commands, discussed later in this chapter. They can also execute programs from either a command procedure file (also called a CPL program) or a command input file (also called a COMINPUT file).

These files allow sequences of PRIMOS commands and subcommands to be written into text files, using an Editor. Invoking these files then executes the commands.

In addition, CPL programs can make run-time decisions about program execution. They can accept arguments, define variables, and execute IF statements, loops, and GOTOS. CPL programs thus offer greater flexibility and power than COMINPUT files.

CPL programs are discussed in Chapter 9. COMINPUT files are discussed in Chapter 10.

### Phantom and Batch Execution

Either CPL programs or special phantom files can execute programs in a phantom environment. Phantoms are discussed in Chapter 10. CPL programs and COMINPUT files can be run as Batch jobs. Batch execution is discussed in detail in Chapter 11.

### EXECUTING SEGMENTED RUNFILES

For programs loaded and saved by SEG, execution is performed at the PRIMOS command level using the SEG command:

SEG pathname

where pathname is the name of a SEG runfile. (SEG looks first for pathname.SEG, then for pathname.) SEG loads the runfile into segmented memory and starts execution. You use SEG to execute runfiles created by SEG's loader; do not use SEG for runfiles created by the R-mode loader. For example:

```
OK, SEG TEST      /* user requests program
THIS IS A TEST    /* output of program
OK,               /* PRIMOS requests next command
```

Upon completion of program execution, control returns to PRIMOS command level.



If both the SEG runfile and the copy of SEG used to invoke it are in memory, then you can usually restart a SEG runfile by giving the command:

S 1000

### EXECUTING R-MODE RUNFILES

For programs loaded in 32R or 64R mode by the loader, execution is performed at the PRIMOS command level using the RESUME command. Command line format for RESUME is:

RESUME pathname

RESUME brings the runfile pathname from the disk into the user's memory, loads the initial register settings, and begins execution of the program. For example:

```
OK, R TEST      /* User requests program
THIS IS A TEST  /* Output of program
OK,             /* PRIMOS requests next command
```

### Note

Do not use RESUME for segmented (64V mode) programs. Use the SEG command (discussed in the first part of this chapter) instead.

When RESUME is invoked, it looks first for pathname.CPL. If pathname.CPL cannot be found, RESUME looks for pathname.SAVE. If pathname.SAVE cannot be found, RESUME looks for pathname without an identifying suffix. For fastest search time, therefore, runfiles should use the .SAVE suffix rather than no suffix.

### The START Command

If a program has been brought into memory (for example, by a previous RESUME command), you can use the START command to initialize the registers and begin execution. Its format is:

START [start-address]

If START is typed without a value for start-address, the program resumes at the address value at which execution was interrupted. To restart the program at a different point, specify an octal starting location as the start-address value. The usual default value for the beginning of FORTRAN programs is 1000.



For example:

```
OK, R TEST1      /* Begin
INPUT NEW KEY: 5 /* Program asks for input
QUIT            /* User hit BREAK to stop
OK, S 1000       /* Restart program from beginning
INPUT NEW KEY:
```

START can also restart a program that has returned control to PRIMOS (for example, because of an error or a FORTRAN PAUSE or CALL EXIT statement).

The applications programmer will almost always use the default forms of the RESUME and START commands (the form discussed here). For a complete treatment of these commands, see The PRIMOS Commands Reference Guide.

Upon completion of the program, control returns to PRIMOS command level.

## RUNTIME ERROR MESSAGES

During program execution, error conditions may be generated and detected by the FORTRAN mathematical functions, file system subroutine calls, or the operating system. A list of run-time error messages is given in Appendix D.

Error messages specific to execution of segmented programs are labeled 64V mode. Some error messages imply system problems beyond the scope of the applications programmer. If so, this is indicated in the explanation of a given error message.



Page 2

1. The purpose of this document is to provide a summary of the information received from the source. The information is classified as CONFIDENTIAL - SECURITY INFORMATION.

2. The information was obtained from a source who has provided reliable information in the past. The information is classified as CONFIDENTIAL - SECURITY INFORMATION.

3. The information is being provided to you for your information only. It is not to be used for any other purpose. The information is classified as CONFIDENTIAL - SECURITY INFORMATION.

4. The information is being provided to you for your information only. It is not to be used for any other purpose. The information is classified as CONFIDENTIAL - SECURITY INFORMATION.

CONFIDENTIAL - SECURITY INFORMATION

5. The information is being provided to you for your information only. It is not to be used for any other purpose. The information is classified as CONFIDENTIAL - SECURITY INFORMATION.

6. The information is being provided to you for your information only. It is not to be used for any other purpose. The information is classified as CONFIDENTIAL - SECURITY INFORMATION.



# 8

## Debugging Programs

### INTRODUCTION

Prime's Source-Level Debugger, DBG, enables programmers writing in FORTRAN IV (V-mode only), FORTRAN 77, Pascal, PL/I Subset G, and RPG II (V-mode) to debug compiled and loaded code using high-level language constructs. DBG (a separately priced product) offers the following features:

### Program Control

<u>Feature</u>	<u>Function</u>
Single Stepping	The debugger offers a comprehensive set of commands to permit single stepping across statements, into and out of called procedures, and so on.
Breakpointing	The user may request that execution be suspended at a specified statement in, entry to, or exit from a procedure.
Action List on Breakpoint	The user may request the execution of one or more debugger commands each time a breakpoint occurs.



## Conditional Breakpointing

The user may specify that a breakpoint is to take place only if a given expression yields a true result.

## Program Restart

The user may at any time restart the program being debugged.

## Transfer of Control

The user may request that control be transferred to a specified statement when program execution is resumed.

## Call Subroutine or Function

The user may call a subroutine or function from debugger command level.

TracingFeatureFunction

## Tracepointing

The user may request that a trace message be output at a specified statement in, entry to, or exit from a procedure.

## Statement Tracing

The user may specify that a trace message is to be output prior to the execution of each statement or each labelled statement.

## Entry/Exit Tracing

The user may specify that a message is to be printed each time any procedure is called or returns.

## Value Tracing

The user may specify that a given variable, or variables, is to be 'watched' and a message printed any time the value of the variable is changed.

## Traceback

The debugger has the capability to print a procedure call/return stack history.



Data Manipulation

<u>Feature</u>	<u>Function</u>
Value of Variable	The debugger can print the value of any scalar, array or structure, as well as an array cross-section.
Expression Evaluation	The debugger evaluates any expression allowed by the source language.
Assignment	The user may modify the value of a variable.
Examine Expression Type	The user can examine the resultant type of an expression (or simple variable).
Built-in (Intrinsic) Functions	All built-in functions available in FORTRAN, PL/I and Pascal are internal to the debugger and may be referenced in an expression.

Miscellaneous

<u>Feature</u>	<u>Function</u>
Source File Examination	The user may examine, but not change, any text file while executing within the debugger.

This chapter offers an introduction to a few basic commands and techniques for debugging programs with DBG. To learn more about using DBG, consult the Source Level Debugger Guide. For a summary listing of DBG commands, see the Loading and Debugging Companion.

In addition, users with a knowledge of assembly language can use Prime's symbolic debuggers, PSD and VPSD. VPSD can be used to debug V-mode programs written in any language, while PSD debugs R-mode programs. The Assembly Language Programmer's Guide contains a full reference text for these debuggers. The Loading and Debugging Companion contains a summary list of their commands.



USING DBGCompiling the Program

Compile the program in 64V mode and include the -DEBUG option on the command line. For example:

```
FTN MYPROGRAM -64V -DEBUG
```

The -DEBUG option causes the compiler to generate the information necessary for the debugger to recognize and manipulate procedure statements and symbols.

Loading the Program

Load the program normally. For example:

```
OK, SEG -LOAD
$ LOAD MYPROGRAM
$ LI
LOAD COMPLETE
$ SAVE
$ QUIT
```

Invoking and Terminating the Debugger

To invoke the debugger, type DBG, followed by the name of the SEG file containing the program to be debugged. For example, to debug MYPROGRAM, enter:

```
DBG MYPROGRAM
```

The debugger reads the program and symbol table from the SEG file into memory and prints an identification message. When the right angle bracket (>), the debugger's prompt character, appears at the left margin of the terminal, the debugger is ready for command input.

Typing QUIT terminates the debugging session and returns to PRIMOS command level.

STARTING PROGRAM EXECUTION

After invoking the debugger, use the RESTART command to start program execution at the beginning (main entry point) of the main procedure. This command also restarts the program from the beginning at any point in the debugging session. To continue from a place where the program has been stopped without going back to the beginning, use the CONTINUE command.



Note

The variables initialized by a FORTRAN DATA statement or PL/I INITIAL clause will not be reinitialized if RESTART is given after execution has begun.

STOPPING EXECUTION

Breakpointing (BREAKPOINT command) and single stepping (STEP command) permit the user to interrupt program execution at a specific place. This may be useful for examining program data. For example, assume a subroutine is suspected of incorrectly handling a variable. Setting breakpoints at the entry to the subroutine and just before returning from the subroutine will permit examination of the variable before and after it has been acted upon by the subroutine.

On the other hand, with a fairly complicated subroutine which is branching incorrectly, it might be useful to stop at the beginning of the routine and single step through it until suspicious behavior starts to occur.

Breakpointing

A breakpoint stops a program when it reaches a specific location. The BREAKPOINT command sets new breakpoints and modifies existing ones. Figure 8-1 shows a listing of a simple FORTRAN program. Figure 8-2 shows the use of the evaluate (:) command with BREAKPOINT to examine certain variables within that program once the breakpoint has been set.

Use the SOURCE command (described later in this chapter) to find the physical source line number to set a breakpoint. (See Figure 8-4 for an example.)



```

(0001)      J = 0
(0002)      K = 0
(0003)      DO 30 I = 1,10
(0004)      J = J + 1
(0005)      K = K + 1
(0006)      CALL TEST (J,K)
(0007)      WRITE (1,20) J,K
(0008) 20    FORMAT (2I5)
(0009) 30    CONTINUE
(0010)      CALL EXIT
(0011)      END

```

PROGRAM SIZE:   PROCEDURE - 000072   LINKAGE - 000040   STACK - 000024  
 0000 ERRORS [<.MAIN.>FIN-REV18.1]

```

C
(0012)  C
(0013)  C
(0014)      SUBROUTINE TEST (J,K)
(0015) 10  J = J + 1
(0016) 20  K = K + 1
(0017)      IF (J .EQ. 5) GO TO 100
(0018)      IF (J .EQ. 6) GO TO 110
(0019) 100  IF (K .EQ. 10) GO TO 200
(0020) 110  IF (K .EQ. 20) GO TO 210
(0021)      GO TO 20
(0022) 200  J = J + K
(0023)      GO TO 300
(0024) 210  J = K
(0025) 300  RETURN
(0026)      END

```

PROGRAM SIZE:   PROCEDURE - 000073   LINKAGE - 000024   STACK - 000032  
 0000 ERRORS [<TEST >FIN-REV18.1]

Sample Program  
 Figure 8-1



```
> BREAKPOINT TEST\25
> RESTART
```

```
**** breakpointed at TEST\25 ($300)    $300 is the FORTRAN
> : J                                statement label
J = 12
```

Basic Breakpoint  
Figure 8-2

#### Note

In Figure 8-2, the backslash character is used to separate the name of a procedure from the statement number or label of a statement within it. Statement numbers correspond to those generated in a compiler listing file and should not be confused with FORTRAN statement labels. The debugger SOURCE command prints a source file with these statement numbers. They are frequently used as breakpoint identifiers, as in this example.

Clearing Breakpoints: The CLEAR command clears a single breakpoint. "CLEARALL procedure-name" clears all breakpoints in procedure-name, while CLEARALL clears all breakpoints in all procedures.

Listing Breakpoints: The LIST command prints the attributes of one breakpoint. "LISTALL procedure-name" lists the attributes of all the breakpoints in procedure-name. "LISTALL" prints the attributes of all the breakpoints in all the procedures.

#### Single Stepping

The STEP command stops program execution after a given number of statements, without referencing specific labels or statement numbers. Figure 8-3 shows the use of STEP to examine a routine in detail.

#### EXAMINING AND MODIFYING DATA

Usually breakpointing and single stepping are used for the examination of data. The evaluate (:) command makes it easy to do this.

The LET command changes the values of variables and can be useful for testing new values before reediting the source program.



```
> BREAKPOINT 15
> CONTINUE
32 20
```

```
**** breakpointed at TEST\15 ($10)
```

```
> : J
J = 33
> : K
K = 21
> STEP
```

```
**** "step" completion at TEST\16 ($20)
```

```
> : J
J = 34
> : K
K = 21
> STEP
```

```
**** "step" completion at TEST\17 ($20+1)
```

```
> : J
J = 34
> : K
K = 22
```

### Single Stepping

#### Figure 8-3

### EXAMINING THE SOURCE CODE

During debugging, you may wish to examine your source code from time to time. The SOURCE command makes this possible. SOURCE examines and displays source code without leaving DBG. It uses a subset of Prime's Editor (ED) commands, plus some of its own. Table 8-1 lists those SOURCE arguments that are taken from ED. Figure 8-4 illustrates the use of the SOURCE command. (For the other arguments to the SOURCE command, see the Source Level Debugger Guide.)

#### Note

The source command may change your current directory. This happens if you give the SOURCE command after your program has set the current attach point, and the source file is in a directory different from the new current attach point.



Table 8-1  
SOURCE Command Arguments (from ED)

Argument	Description
<u>T</u> OP	Position line pointer to top of file.
<u>B</u> OTTOM	Position pointer to bottom of file.
<u>B</u> RIEF	Don't print target lines of FIND, LOCATE, POINT and NEXT operations.
<u>V</u> ERIFY	Print target lines of FIND, LOCATE, POINT and NEXT operations.
<u>P</u> RINT	Print line(s).
<u>W</u> HERE	Print current line number.
<u>P</u> OINT	Position to specific line.
<u>N</u> EXT	Move line pointer forward or backward.
<u>M</u> ODE	Set edit mode; the only mode implemented is NUMBER/NUMBER.
<u>L</u> OCATE	Locate line with the specified text string.
<u>F</u> IND	Locate line with the specified text string beginning in a given column.



\*\*\*\* breakpointed at \$MAIN\1

> SOURCE PRINT 30

```
1:      J = 0
2:      K = 0
3:      DO 30 I = 1,10
4:      J = J + 1
5:      K = K + 1
6:      CALL TEST (J,K)
7:      WRITE (1,20) J,K
8: 20    FORMAT (2I5)
9: 30    CONTINUE
10:     CALL EXIT
11:     END
12: C
13: C
14:     SUBROUTINE TEST (J,K)
15: 10    J = J + 1
16: 20    K = K + 1
17:     IF (J .EQ. 5) GO TO 100
18:     IF (J .EQ. 6) GO TO 110
19: 100   IF (K .EQ. 10) GO TO 200
20: 110   IF (K .EQ. 20) GO TO 210
21:     GO TO 20
22: 200   J = J + K
23:     GO TO 300
24: 210   J = K
25: 300   RETURN
26:     END
BOTTOM
```

Debugger SOURCE Command  
Figure 8-4



SAMPLE PROGRAM DEBUGGING SESSION

Consider the following (undebugged) FORTRAN program, SQUARES.FTN, which is used as an example in the text below:

```

(0001) C   Print the squares of the numbers 1 through 10.
(0002) C
(0003)      DO 100 I = 1, 10
(0004)      J = SQUARE (I)
(0005)      WRITE (1, 80) J
(0006) 80   FORMAT (I5)
(0007) 100  CONTINUE
(0008)      CALL EXIT
(0009)      END
(0010)
(0011)      INTEGER FUNCTION SQUARE (I)
(0012)      SQUARE = I**2
(0013)      RETURN
(0014)      END

```

Assuming that this program resides in a file called SQUARES, it would be compiled with the following PRIMOS command:

```

OK, FTN SQUARES -64V -DEBUG
0000 ERRORS [<.MAIN.>FTN-REV19.0]
0000 ERRORS [<SQUARE>FTN-REV19.0]

```

OK,

The -DEBUG argument informs the FORTRAN compiler that the program to be compiled will later be debugged using DBG.

The sequence of commands used to load the program is identical to that used without DBG, namely:

```

OK, SEG -LOAD
$ LOAD SQUARES
$ LIBRARY
LOAD COMPLETE
$ SAVE
$ QUIT

```

OK,



There now exists a SEG run-file, SQUARES.SEG, containing a program which has been compiled and loaded without errors. The first attempt at execution fails, producing an output which looks like this:

OK, SEG SQUARES

0  
0  
0  
0  
0  
0  
0  
0  
0  
0  
0

OK,

The user now enters the debugger using the following PRIMOS command:

OK, DBG SQUARES

\*\*Dbg\*\* revision 19.1h (05-November-1981)

>

The user decides to place a breakpoint following the call to function SQUARE in order to look at the argument and returned value. He enters:

> BREAKPOINT 5

>

This breakpoint command will cause the debugger to regain control immediately prior to the execution of the statement on source line 5, the FORTRAN WRITE statement.

The user then begins program execution with the debugger's RESTART command. This is followed by the occurrence of the breakpoint. The interaction looks like this:

> RESTART

\*\*\*\* breakpointed at \$MAIN\5

>

The breakpoint message says, effectively, "I have encountered the breakpoint which you set on source statement number 5 in the FORTRAN main program." The prompt character indicates that the debugger is once again at command level awaiting a command.



The user may now inspect the values of variables I and J:

```
> : I
I = 1
> : J
J = 0
```

If you precede any variable name or expression with a colon (:) or space, the value of the variable or the resultant value of the expression will be printed on the user's terminal.

The user sees that the value of I is correct (I is the control variable for the do loop; this is the first iteration through the loop), however the value of J is in error. The user decides to suspend program execution earlier in the loop, in function SQUARE immediately after the function value is computed. To breakpoint at line 13 of the function SQUARE, he enters:

```
> BREAKPOINT SQUARE\13
> CONTINUE
0
```

\*\*\*\* breakpointed at SQUARE\13

(The "0" which appears above is output by the FORTRAN WRITE statement on source line 5.)

The user chooses to look at the value of the argument to the function and that of the computed returned value:

```
> ARGUMENTS
I = 2
> : SQUARE
SQUARE = 4
>
```

The user finds that both the argument and function values are correct. He continues execution until the breakpoint on source statement 5 in the main program occurs by entering:

```
> CONTINUE
```

\*\*\*\* breakpointed at \$MAIN\5

```
>
```



Once again looking at the returned value,

```
> : J
J = 0
>
```

the user finds it incorrect. He suspects that the data types of the SQUARE function mismatch between the FORTRAN main program and SQUARE. This is confirmed by typing:

```
> TYPE $MAIN\SQUARE
entry constant (real*4 function)
> TYPE SQUARE\SQUARE
integer*2 static
>
```

The user sees that the data types do indeed mismatch (REAL\*4 vs. INTEGER\*2). Upon correcting, recompiling, and reloading the program, he finds that it works:

OK, DBG SQUARES.SEG

**\*\*Dbg\*\*** revision 19.1h (05-November-1981)

> RESTART

```
1
4
9
16
25
36
49
64
81
100
```

EXIT. Program exit from \$MAIN\8 (\$100+1).

> QUIT

OK,

The EXIT message indicates that the program called the system subroutine EXIT from source line 8 in the FORTRAN main program. (The "\$100+1" alternatively identifies the statement as the one following FORTRAN statement label 100.)

The QUIT command causes the debugger to exit to PRIMOS command level.



# 9

## The Basics of CPL

### WHAT IS CPL?

CPL is Prime's "Command Procedure Language" — a high-level language that operates at PRIMOS command level. CPL provides:

- Variables
- Function calls
- Branching (via such directives as &IF...&THEN...&ELSE, &GOTO, &SELECT)
- Error handling and debugging facilities

### LEARNING CPL

CPL provides features for users who want maximum ease of use for simple programs, and for users who want maximum power and flexibility. This chapter provides a brief overview of CPL, and an introduction to the major "ease of use" features. The CPL User's Guide provides full tutorial and reference information for all CPL features.



HOW DOES CPL WORK?

CPL has two parts: the language and the interpreter. The CPL language allows users to write CPL programs which contain either a sequence of PRIMOS commands or a combination of PRIMOS commands and CPL directives. The commands give instructions to PRIMOS, or to one of its subsystems. The directives give instructions to the CPL interpreter itself. (PRIMOS never sees these directives; it sees only the commands which the interpreter passes to it.)

When the programs are executed, the CPL interpreter first evaluates variables and function calls and replaces them with their correct values. It then interprets and acts upon CPL directives. Finally, it passes the resulting commands to PRIMOS for execution. (Figure 9-1 illustrates the evaluation of a CPL directive and the resulting execution of a command.) Thus, a lengthy series of commands can be set in motion by a single command, relieving the user of much repetitive typing; yet run-time decisions can be made at any time during the file's execution.

CREATING AND EXECUTING CPL PROGRAMS

Like other high-level language programs, CPL programs are written using the editor (ED). Their format is simple, being based on the principle of one statement per line. Indentation may be used as desired, for ease of reading. (Format rules are explained where applicable throughout this chapter. They are explained fully in Chapter 3 of the CPL User's Guide.)

Note

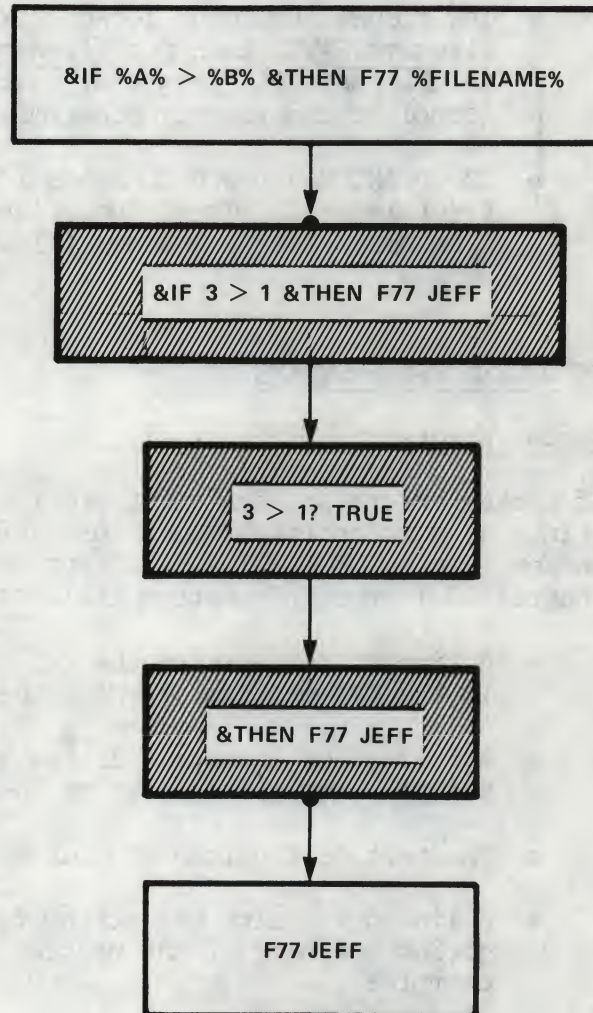
CPL programs must be given names that end with the .CPL suffix.

CPL programs are not compiled or loaded. As soon as they have been written, they are ready to execute.

You may run CPL programs interactively with either the CPL command or the RESUME command. You may also run them as phantoms (with the PHANTOM command), or as Batch jobs (with the JOB command). For details of how to run CPL programs as Batch jobs, see Chapter 11.



1. CPL file contains the statement:
2. The CPL interpreter reads the statement, substituting current values for variable references:
3. The CPL interpreter tests:
4. Since the test condition is true, CPL executes the &THEN statement, passing the command "F77 JEFF" to the Standard Command Processor:
5. Command processor executes the command:



Execution of a Sample CPL Directive  
Figure 9-1



You do not need to specify the .CPL suffix when you submit a CPL program for execution. The CPL, RESUME, JOB, and PHANTOM commands will all look for filename.CPL when you specify filename. Their search rules are as follows:

- The CPL command looks first for filename.CPL, then for filename. It runs either one as a CPL program.
- The RESUME command looks first for filename.CPL, then for filename.SAVE, then for filename. It runs files whose names end in CPL as CPL programs. It runs all other files as run-time (compiled and loaded) programs.
- The PHANTOM and JOB commands look first for filename.CPL, then for filename. They run files whose names end in .CPL as CPL programs; they run other files as command input files.

## DEBUGGING CPL PROGRAMS

### Syntax Errors

If syntax errors prevent a CPL program from executing, the interpreter prints a substantial amount of information at the user's terminal and/or into a COMOUTPUT file. (For details on COMOUTPUT files, see Chapter 10.) The information includes:

- A line of text giving the CPL error number and the line number in the CPL program at which the error occurred.
- A full error message. If the text containing the error can be printed, it will be part of the message.
- The text of the line of source code in which the error occurred.
- A line describing the action taken by the CPL interpreter and giving the name of the program in which the error occurred. For example:

OK, R BLUNDER

CPL ERROR 40 ON LINE 2.

A reference to the undefined variable "FILENAME" has been found in this statement.

SOURCE: como %filename%.como

Execution of procedure terminated. BLUNDER (cpl)  
ER!



In this example, the program BLUNDER.CPL contained a misprint, FILLNAME, for the variable, FILENAME.

If CPL programs are halted by PRIMOS errors, then the normal PRIMOS error message (ending in "ER!") is printed out.

#### Note

Either CPL or PRIMOS syntax errors halt the CPL program and return the user to command level. However, warning messages from PRIMOS or its subsystems will not normally halt execution of a CPL program.

CPL's &SEVERITY directive (explained in the CPL User's Guide) allows users to modify a program's response to PRIMOS errors and warnings.

#### Logic Errors

If a CPL program runs, but produces erroneous results, you can use the facilities provided by CPL's &DEBUG directive to track down the errors. CPL's debugging facilities offer:

- Variable watching, to print out the value of a variable each time the value is set or altered.
- Echoing, to print commands and directives as they are read. (This can tell you if unexpected branching is occurring.)
- A no-execute option, to allow the interpreter to "walk through" the CPL program without actually executing any of the commands it contains.

For full details on debugging CPL programs, see Chapter 10 of the CPL User's Guide.

#### USING PRIMOS COMMANDS IN CPL PROGRAMS

The simplest CPL programs are composed entirely of PRIMOS commands. For example, a CPL file might execute three programs. Such a program might be named RUNNEM.CPL. It might look like this:

```
RESUME NEW_TALLY
RESUME NEW_SORT
SEG DAILY_UPDATE
```



Which PRIMOS Commands Can you Use?

CPL files that consist entirely of PRIMOS commands can use the following commands:

- All compiler commands: COBOL, F77, FTN, PL1G, PMA, RPG, etc.
- All commands which execute programs. For example:

```
SEG THISFILE.SEG
R THATFILE.SAVE
R FILE.CPL
BASICV ANYFILE
```

- Any user commands which do not invoke a subsystem or initiate a dialog. For example, you may use:

```
ATTACH
LD
CREATE
COPY
DELETE
CNAME
PASSWD
SIZE
```

- Commands that invoke interactive subsystems or user programs, if the user is going to supply the data or subcommands from the terminal at runtime. For example:

```
ED
SEG
SORT
```

If you want the CPL program itself to supply the data or subcommands, you must use CPL's &DATA directive, explained later in this chapter.

What Commands Can't You Use?

Do not use the commands:

- COMINPUT (in any form)
- CLOSE -ALL
- DELSEG ALL

in a CPL file. Any of these commands will abort execution of the file.



CPL DIRECTIVES

The CPL language contains the following directives. Those marked with asterisks are discussed in the remainder of this chapter. All are discussed in detail in the CPL User's Guide.

DirectiveUseVariable-handling and Argument-handling Directives

- |            |  |
|------------|--|
| * &ARGS    | Defines names (plus types and default values, if desired) for arguments to be passed to the CPL program from the command line that executes the program. |
| * &SET_VAR | Defines a variable and sets its value; or alters the value of an existing variable.  |

Branching Directives

- |                       |   |
|-----------------------|---|
| * &IF...&THEN...&ELSE | Allows conditional branching, choosing between Boolean (TRUE-FALSE) alternatives.   |
| &SELECT               | Allows conditional branching among a number of specified alternatives.  |
| * &DO...&END          | Groups statements to be treated as a single unit syntactically. (For example, a "DO GROUP" may represent the action to be taken by a &THEN or &ELSE directive.)   |
| &DO iteration...&END  | Allows conditional iteration (that is, repeated execution) of a group of statements. CPL supports counted loops, &WHILE, &UNTIL, and &REPEAT loops. It also has two types of loops that take advantage of CPL's "wild card" capabilities. |
| * &DATA...&END        | Groups statements to be treated as data or subcommands for user programs or PRIMOS utilities (such as ED or FUTIL).   |
| &GOTO...&LABEL        | &GOTO forces an unconditional branch to the statement immediately following the &LABEL directive.   |



- \* **&RETURN** Halts execution of program or routine and returns control to user or calling program. The CPL interpreter puts an implicit **&RETURN** statement at the end of each CPL program. The **&RETURN** directive can also pass messages and/or integer severity codes to the user or caller of the halted program or routine.
- &STOP** Halts execution of a CPL program, whether it is used in the main program or in an internal routine. The **&STOP** directive can also pass messages and/or integer severity codes to its program's caller.

### Subroutines and User-defined Functions (Internal and External Procedures)

- &CALL** Calls (transfers control to) an internal routine.
- &ROUTINE** Defines and names an internal routine.
- &RESULT** Allows a CPL program to serve as a user-defined function for other CPL programs.

### Execution-Control Directives

- &DEBUG** Turns on (or off) CPL's debugging facility during program execution. Options to these directives specify debugging actions to be taken.
- &EXPAND** Allows use of specified ABBREV file by the CPL program. (For details on ABBREV files, see Chapter 17.)
- &SEVERITY** Defines the behavior of the CPL program (stop, continue, or call an error-handling routine) when system-defined errors or warnings occur.

### Error-handling and Condition-handling Directives

- &CHECK...&ROUTINE** Checks for user-defined error conditions. Defines an internal routine to act as error-handler if the error occurs.
- &ON...&ROUTINE** Defines a routine to act as a condition handler for a CPL program or routine. (See Chapter 20 for information on conditions and on PRIMOS's Condition Mechanism.)



<code>&amp;REVERT</code>	Disables a specified condition handler.
<code>&amp;SIGNAL</code>	Signals a user-defined (or system-defined) condition to the condition mechanism.

### USING VARIABLES IN CPL PROGRAMS

There are three ways in which CPL programs can obtain variable data:

- The variable data can be passed to the CPL programs as arguments when the CPL program is invoked. In this case, the `&ARGS` directive is used inside the CPL program to define variable-names for the arguments and to match these names to the data supplied in the invocation.
- Variables may be defined and assigned values inside the CPL program by using the `&SET_VAR` directive.
- A user can maintain a global variable file. (See Chapter 17 for details.) By using the command "DEFINE\_GVAR filename" in a CPL program, the user allows the program to access that file and reference the variables contained in it.

Once a variable has been defined in a CPL program (by the `&ARGS` or `&SET_VAR` directives, or by the `DEFINE_GVAR` command), it may be referenced by placing its name inside percent signs. Thus, if `NAME` was the name of the variable, `%NAME%` would be a reference to that variable. When the CPL interpreter read the reference, it would substitute the current value of `NAME` for the string `%NAME%` in the command line or directive in which the reference occurred.

### Using the &ARGS Directive

The simplest form of the `&ARGS` directive is:

```
&ARGS variable_name [;...variable_name]
```

For example, a CPL program (named `F7.CPL`) that compiles any `F77` source file might read:

```
&ARGS FILENAME
COMO %FILENAME%.COMO
DATE
F77 %FILENAME% -DEBUG
COMO -E
```



In this example, the &ARGS directive defines one variable, FILENAME. When the file is invoked, the name of the file to be compiled is supplied as an argument, following the name of the CPL file. For example, the invocation might read:

```
R F7 JEFF
```

In this example, the &ARGS directive takes the character string JEFF and assigns it to the variable FILENAME. JEFF is now the value of FILENAME.

From now on, each time a variable reference, %FILENAME%, is found, the CPL interpreter substitutes the character string JEFF for the character string %FILENAME%. Thus, the command:

```
COMO %FILENAME%.COMO
```

becomes

```
COMO JEFF.COMO
```

while the command:

```
F77 %FILENAME% -DEBUG
```

becomes

```
F77 JEFF -DEBUG
```

Note that the variable, FILENAME, is not enclosed in percent signs when it is being defined in the &ARGS directive, but is enclosed in percent signs whenever it is "referenced"—that is, whenever its value, rather than its name, is wanted.

#### Note

When a variable reference is juxtaposed to another character string, with no blanks between them (as in %FILENAME%.COMO), the value of the variable is concatenated with the other string, (as in JEFF.COMO). Two or more variable references may also be juxtaposed, (as in %FILENAME%%FILENAME%). Again, a single string results (JEFFJEFF).

#### Multiple Arguments

When multiple arguments are given, the variable names in the &ARGS directive must be separated by semicolons. For example:

```
&ARGS FILENAME; COMPILER
```



Now you can write a more general CPL file, called `COMPILE_ALL.CPL`, that can compile `FTN`, `F77`, or `PL1G` source files. It reads:

```
&ARGS FILENAME; COMPILER
COMO %FILENAME%.COMO
DATE
%COMPILER% %FILENAME% -64V -DEBUG
COMO -E
```

Invoking this file by typing:

```
R COMPILE_ALL FRED FTN
```

creates the command:

```
FTN FRED -64V -DEBUG
```

In general, arguments are defined by their position in the command line. In the above example, the first argument, "FRED", became the value of the first variable in the `&ARGS` line, "FILENAME". The second argument, "FTN", was assigned to the second variable, "COMPILER". Giving the arguments in reverse order:

```
R COMPILE_ALL FTN FRED
```

would assign "FTN" to "FILENAME" and "FRED" to "COMPILER".

### Omitted Arguments

If an argument is omitted from the command line, the CPL interpreter sets its value to the explicit null string, `''`. The PRIMOS command processor then removes the null string before executing the command. In the above example, the command:

```
R COMPILE_ALL TESTFILE
```

assigns the value `TESTFILE` to the variable `FILENAME`, and assigns the null string to the variable `COMPILER`. The resulting PRIMOS command first becomes:

```
'' TESTFILE -64V -DEBUG
```

and then becomes:

```
TESTFILE -64V -DEBUG
```

Since PRIMOS can do nothing worthwhile with such a command, it returns you to command level with no compilation having occurred.



Note

CPL offers several ways to deal with null arguments. These are explained in the CPL User's Guide.

The &SET\_VAR Directive

The form of the &SET\_VAR directive is

```
&SET_VAR name := value
```

For example:

```
&SET_VAR A := AMY
```

defines the variable A and gives it the value AMY.

value may also be an expression. For example:

```
&SET_VAR X := 10
&SET_VAR Y := 5
&SET_VAR Z := %x% + %y%
```

These three directives define the variables X, Y, and Z. X has the value of 10, Y the value of 5, and Z the value of 15.

Note

In CPL programs, all operators MUST be separated from their operands by one or more spaces.

DECISION-MAKING (BRANCHING) IN CPL PROGRAMS

When a CPL program contains only PRIMOS commands (or PRIMOS commands plus variables), it is executed sequentially; that is, each command (each line of the program) is executed in turn.

Sometimes, however, you may want to alter the sequence in which the commands are executed. To alter the "flow of control" in this way, you use CPL's flow of control directives. The simplest and most important of these is the &IF directive.



The &IF Directive

The form of the &IF directive is:

&IF test &THEN statement

test is a logical test which can be answered TRUE or FALSE (for example, &IF A = B, &IF %NUMBER% < 10). statement is either a command or a CPL directive.

test may test variables, constants, functions or expressions against each other. For example:

&IF %A% = 10	(variable and constant)
&IF %A% < %B%	(two variables)
&IF %A% < %B% + %C%	(variable and expression)
&IF %A% + %B% = %D% + 30	(two expressions)
&IF [LENGTH %A%] < 100	(function and constant)

How the &IF Directive Works: When the CPL interpreter reads an &IF directive, it substitutes current values for any variable references, expressions, or function calls it finds. Then it tests to see if test is true or false. If test is true, the interpreter executes the command or directive that forms the &THEN statement.

For example, suppose you compile a program frequently, but only occasionally want to spool the listing file. You could use an argument and the &IF directive to tell the CPL program whether or not to spool the listing file. Here is a program to do it (called CNS.CPL):

```
&DEBUG &ECHO COM
/*This program compiles and optionally spools
/*an F77 program.
/*Give the argument "SP" to spool the listing file.
&ARGS FILENAME; SP
/*Open the COMOUTPUT file and compile the program
COMO %FILENAME%.COMO
DATE
F77 %FILENAME% -L %FILENAME%.LIST -XREF
/*If desired, spool it.
&IF %SP% = SP &THEN SPOOL %FILENAME%.LIST -AT MS3
COMO -E
```

Note

As this program shows, you can use /\* to place comments in CPL programs.



If you give the command:

```
R CNS JEFF SP
```

then the test, `SP = SP`, is true, and the listing file, `JEFF.LIST`, is spooled. If you give the command:

```
R CNS JEFF
```

the test is false (the null string does not equal "SP"). In this case, the listing file is not spooled. Instead, the CPL interpreter ignores the `&THEN` statement, and passes on to the next line in the program (in this case, `"COMO -E"`).

### The &ELSE Directive

The `&IF` directive may be used by itself, as in the `CNS.CPL` sample program above; or it may be followed by the `&ELSE` directive. When used by itself, `&IF` tells the interpreter either to execute or to ignore some statement. (In the example, spool the file, or don't spool it.) When the `&IF` and `&ELSE` directives are used together, they tell the interpreter to choose between two courses of action.

The form of the paired directives is:

```
&IF test &THEN statement-1
      &ELSE statement-2
```

If test is TRUE, statement-1 is executed. If test is false, statement-2 is executed. For example, suppose you compile many `F1N` files and a few `F77` files. You might want a program (called `COMPILE2.CPL`) that looked like this:

```
&ARGS FILENAME; COMPILER
&IF %COMPILER% = F77 &THEN F77 %FILENAME% -DEBUG -32I
&ELSE F1N %FILENAME% -64V
```

If you give the command `"R COMPILE2 THISFILE F77"`, the test (`F77 = F77`) becomes true, and `THISFILE` is compiled by the `F77` compiler. If you give any other value for the "compiler" argument—or if you omit that argument altogether—`THISFILE` is compiled by the `F1N` compiler.

### Nested &IFs

`&IF` directives may be nested: that is, either the `&THEN` or the `&ELSE` action of one `&IF` directive may be another `&IF` directive. Nested `&IF` statements are discussed in the CPL User's Guide.



&DO GROUPS

In the examples above, the &THEN and &ELSE directives execute single commands. These directives may also execute groups of commands, by using the &DO and &END directives to mark the beginning and end of the command groups.

The format for &DO groups is as follows:

```
&DO
    statement 1
    statement 2
    .
    .
    .
    statement n
&END
```

Normally, each statement in a CPL program represents one action the interpreter is asked to perform. In a &DO group, however, all the statements between the &DO and the &END represent a single action to the interpreter. Thus instead of saying :

```
&IF test &THEN statement-1
      &ELSE statement-2
```

we can say:

```
&IF test &THEN &DO
    first-group-of-statements
    &END
    &ELSE &DO
    second-group-of-statements
    &END
```

For example:

```
&ARGS %MONTH%
&IF %MONTH% = DEC &THEN &DO
    SEG MONTHLY_REPORT
    SEG END_OF_YEAR_REPORT
    SEG XMAS_LIST
    &END
&ELSE SEG MONTHLY_REPORT
```



USING FUNCTIONS IN CPL PROGRAMS

Like other high-level languages, CPL provides built-in functions to simplify frequently made tests and computations. Functions appear in CPL files in the form of function calls; that is, functions and their arguments are enclosed in square brackets ([FUNCTION arg]). When a function call appears in a command or directive, the CPL interpreter performs the required test or computation, and substitutes the character string thus produced for the character string represented by the function call.

The NULL function

One of the most useful CPL functions is the NULL function. Its form is as follows:

```
[NULL var]
```

where var is any CPL variable.

The NULL function tests for a null character string, returning the character string TRUE if it finds one and the character string FALSE if it does not. Since the value of an omitted argument is the null string, the NULL function can be used in &IF directives to test for an omitted argument.

For example, a test for a null argument might be used to set the home UFD for some procedure. Such a CPL program might begin:

```
&ARGS WHERE
IF [NULL %WHERE%] &THEN ATTACH MY_UFD
  &ELSE ATTACH %WHERE%
```

Specifying WHERE allows you to make any desired ATTACH; omitting WHERE attaches you to your default choice (MY\_UFD).

The EXISTS Function

The EXISTS function is a Boolean function that determines

- Whether or not a file system object exists.
- Whether it matches a specified type (file, directory, segment directory, or access category).

The form of the function call is:

```
[EXISTS pathname [type]]
```

pathname is the name or pathname of a file system object.



type is one of the following:

- ANY
- FILE
- DIRECTORY or -DIR
- SEGMENT\_DIRECTORY or -SEGDIR
- ACCESS\_CATEGORY or -ACAT

If type is present, then the EXISTS function returns the value TRUE if pathname does exist and is of the right type. It returns the value FALSE if pathname does not exist or if it is of the wrong type. If type is not present, the EXISTS function merely reports whether pathname exists or not (that is, -ANY is the default when type is omitted). The following examples illustrate the use of the EXISTS function.

The first example checks to see if a "new" file has been written. If so, the program calls ED to allow its user to edit the new file. If the new file does not exist, the program calls ED to allow the user to edit the older version:

```
&IF [EXISTS MEMO.NEW] &THEN ED MEMO.NEW
    &ELSE ED MEMO
```

The second example uses the "NOT" symbol, a caret (^), to reverse the value returned by EXISTS. This program wants to attach to a specific directory. If the directory doesn't exist, the program will create it before doing the ATTACH:

```
&IF ^ [EXISTS SUBDIR] &THEN CREATE SUBDIR
ATTACH *>SUBDIR
```

#### USING CPL WITH SUBSYSTEMS: &DATA GROUPS

Many of Prime's utilities, such as ED (the text editor) and SEG (the V-mode and I-mode loader), require subcommands to accomplish their function. Similarly, many user programs require that data be typed in from the terminal. CPL's &DATA directive allows CPL programs to supply the data or subcommands needed by these programs and utilities.

&DATA groups resemble &DO groups in that both are groups of statements set off by an opening directive (&DO, &DATA), and a closing &END. In each case, the statements within the group are treated as a unit.



The form of the &DATA group is:

```
&DATA command
Statement-1
Statement-2
.
.
.
Statement-n
&END
```

Command is the command that invokes the subsystem or utility; for example: "&DATA ED filename".

Statement 1 through statement-n represent the commands or data to be passed to the subsystem or user program. As with all CPL statements, they may include variables, function calls, and directives.

The &END statement, on a line by itself, ends the &DATA group.

Here is an example of a CPL program (here named CLR.CPL) that compiles, loads, and executes a PL/I Subset G program:

```
/*CPL program to compile, load, and execute a PLI G program
/*usage: R CLR FILENAME
/*
&ARGS FILENAME
PLIG %FILENAME%          /*Compile program
/*
&DATA SEG -LOAD          /*Invoke SEG
LOAD %FILENAME%          /*Provide SEG commands
LI PLIGLB                /*via &data directives
LI
SA
QU
&END                      /*end of &data group
SEG %FILENAME%.SEG        /*execute run-file
```

### Terminal Input in &DATA Groups

Sometimes you may want a CPL file to invoke a subsystem or user program, give a few subcommands from within the CPL file, and then allow you to give further commands from your terminal. You do this by including CPL's &TTY directive within the &DATA group.



The format is:

```

DATA
statement-1
.
.
statement-n
TTY
END

```

The `&TTY` directive executes after all other statements in the `&DATA` group have been executed. When the `&TTY` directive does execute, control returns to the user at the terminal. When the user leaves the subsystem, control returns to the CPL file.

#### Using &TTY in a Program

This example shows how the `&TTY` directive might work with a user program. Assume a program (named `PURCHASE`) that asks for five items of information about a customer purchase:

```

Dept. name:
Dept. number:
Customer name:
Acct. number:
Amount of purchase:

```

A given department (for instance, the hardware department) might use a CPL program (named `UPDATE.CPL`) to invoke the `PURCHASE` program and pass it its first two items of information. The statements would look like this:

```

DATA R PURCHASE
  HDWR
  38
TTY
END

```

The example as shown could be a complete CPL program. Or, it might be part of a larger program.



A terminal session might look like this:

```
OK, R UPDATE
dept. name: HDWR
dept. number: 38
customer name: H.L. Smith
acct. number: 35684
amount of purchase: 536.89
OK,
```

#### Note

By using a loop and the RESPONSE function, you could write a CPL program that would pass information for any number of purchases to program PURCHASE. The CPL User's Guide explains how to do this.

#### HOW CPL PROGRAMS END: THE &RETURN DIRECTIVE

Every CPL program ends with the directive &RETURN. You may either supply this directive as the last line of the CPL file or may allow the CPL interpreter to add the directive at the file's end.

You may also use the &RETURN directive to stop the program before the end of the file. For example:

```
&ARGS A
.
.
.
&IF %A% > 20 &THEN &RETURN
&ELSE &DO
.
.
.
.
.
&END
&RETURN
```



# 10

## Command Files and Phantoms

### INTRODUCTION

This chapter discusses:

- How to create and run COMINPUT files
- How to create COMOUTPUT files
- How to use DATE, TIME, and RDY in command files
- How to run phantoms

Batch execution of command files will be discussed in Chapter 11.

### COMMAND FILE REQUIREMENTS

Command input files may contain any legal PRIMOS commands, utility subcommands, or dialog responses, on a line-for-line basis (i.e., each line in the file must correspond to a line as it would be typed at a terminal.) Each utility except Batch imposes certain requirements:

- For COMINPUT, the last command should be COMINPUT -TTY or COMINPUT -END.
- For PHANTOM, the last command should be LOGOUT.
- For Batch, any command file can be used.



Comments

You can document command input files by including comment lines at PRIMOS command level. A line beginning with a slash and asterisk, (/\*), is interpreted as a comment and is ignored by PRIMOS. If a command output file is open, any comments entered at the terminal by the user or from a command file are written into the command output file. Any character may be used in a comment line. A comment may also be appended to a command at PRIMOS command level as in:

```
SLIST BENCH07.MAP          /* PRINT MAP FILE
```

THE COMINPUT COMMAND

The COMINPUT command causes PRIMOS to read input from a specified command file rather than from the terminal. Commands are executed as if they were entered at the terminal. The format is:

COMINPUT [command-file] [options] [file-unit]

<u>Variable</u>	<u>Description</u>
<u>command-file</u>	The pathname of the file from which input is to be read.
<u>options</u>	Specify command control flow as detailed below.
<u>file-unit</u>	The PRIMOS file unit number on which the input file is to be opened. If omitted, file unit 6 is used. File units must be octal (i.e., decimal 8 is entered as 10). File unit numbers are necessary for nested command input files.

<u>Options</u>	<u>Description</u>
<u>-TTY</u>	Either one switches the command input stream to the user terminal and closes the command input file.
<u>-END</u>	
<u>-PAUSE</u>	Switches command input stream to the user terminal but does not close the command input file.
<u>-CONTINUE</u>	Returns control to command input file following a CO -PAUSE or an error.
<u>-START</u>	Resumes command following a BREAK interruption of execution of a command file.



The -TTY, -END and -PAUSE options are used only within command files. The -CONTINUE option may be used within command files or typed by the user; the -START option is typed by the user.

The -TTY or -END option should be the final command in the command file (or in the last command file, if files are chained as discussed later in this chapter).

A simple command file, TEST.COMI, might be created to compile the program TEST.FIN:

```
/*BEGIN TEST OF COMMAND FILE
COMOUTPUT TEST.COMO
DATE
/*COMPILE THE PROGRAM IN 64V MODE
FIN TEST -64V
/*LOAD THE PROGRAM
SEG -LOAD
LO TEST
LI
SA
MAP LOADTEST.MAP 7
MAP UNSATISFIED.MAP 3
QU
/*COMMAND FILE TEST COMPLETED
DATE
COMO -END
CO -END
```

The command file would be executed by the command:

```
CO TEST.COMI
```

and would produce the following output file:

```
OK, DATE
11 Feb 82 14:40:00 Thursday
OK, /*COMPILE THE PROGRAM IN 64V MODE
OK, FIN TEST -64V
0000 ERRORS [<.MAIN.>FIN-REV19.0]
OK, /*LOAD THE PROGRAM
OK, SEG -LOAD
[SEG rev 19.0]
$ LO TEST
$ LI
LOAD COMPLETE
$ SA
$ MAP LOADTEST.MAP 7
$ MAP UNSATISFIED.MAP 3
$ QU
OK, /*COMMAND FILE TEST COMPLETED
OK, DATE
11 Feb 82 14:40:16 Thursday
OK, COMO -END
```



Chaining Command Files

The -CONTINUE option of COMINPUT allows you to chain command files. The following example illustrates the chaining of three command files, and shows how file unit conflicts can be avoided. The command file GO.COMI contains the following commands:

```
/* COMPILE THE PROGRAM IN 64V MODE
FIN TEST -64V
/* LOAD THE PROGRAM
COMINPUT LOADTEST.COMI 7
CLOSE 7
/* RETURN COMMAND TO USER TERMINAL
COMINPUT -TTY
```

The command file LOADTEST.COMI contains the following commands:

```
/* LOADTEST COMMAND FILE
SEG -LOAD
LO TEST
LI
SA
QU
COMINPUT MAPS.COMI 10
CLOSE 10
COMINPUT -CONTINUE
```

The command file MAPS.COMI contains the following commands:

```
/* GET FULL MAP AND UNSATISFIED REFERENCES
SEG
VLOAD * TEST
MAP LOADTEST.MAP 7
MAP UNSATISFIED.MAP 3
QU
/* RETURN TO 'CALLING' COMMAND FILE
COMINPUT -CONTINUE 7
```

Typing COMINPUT GO.COMI causes PRIMOS to read and execute the commands in GO.COMI. When the command COMINPUT LOADTEST.COMI 7 is reached, control passes to LOADTEST.COMI, which loads the object file, then calls MAPS.COMI (on file unit '10) to obtain two load maps. When the command COMINPUT -CONTINUE is reached in MAPS.COMI, control returns to the statement following the call in LOADTEST.COMI, which closes the file unit used for MAPS.COMI. When COMINPUT -CONTINUE is reached in LOADTEST.COMI, control similarly returns to GO.COMI. Finally, the command COMINPUT -TTY in GO.COMI returns control to the user's terminal.



```

OK, CO GO.COMI
OK, /*COMPILE THE PROGRAM IN 64V MODE
FIN TEST -64V
0000 ERRORS [<.MAIN.>FIN-REV18.1]
OK, /*LOAD THE PROGRAM
OK, COMINPUT LOADTEST.COMI 7
OK, /*LOADTEST COMMAND FILE
OK, SEG -LOAD
[SEG rev 19.0]
$ LO TEST
$ LI
LOAD COMPLETE
$ SA
$ QU
OK, COMINPUT MAPS.COMI 10
OK, /*GET FULL MAP AND UNSATISFIED REFERENCES
OK, SEG
[SEG rev 19.0]
#VLOAD * TEST
$ MAP LOADTEST.MAP 7
$ MAP UNSATISFIED.MAP 3
$ QU
OK, /*RETURN TO 'CALLING' COMMAND FILE
COMINPUT -CONTINUE 7
OK, CLOSE 10
OK, COMINPUT -CONTINUE
OK, CLOSE 7
OK, /*RETURN COMMAND TO USER TERMINAL
OK, COMINPUT -TTY
OK,

```

### Errors

If the command file encounters an error from which it cannot recover, it returns input control to the terminal, leaving the command file open. The user may type a correct version of the offending line, and then resume input from the command file by the command:

```
CO -CONTINUE [file-unit]
```

If a file-unit number is not given, PRIMOS will continue the command input file open on file unit 6.



Closing Command Input Files

In chaining command files, the 'called' files should be closed upon returning to the 'calling' files, either by file unit number (as in the example above) or by filename. You should make certain that the file units to be used for the command input files are not already opened (or going to be opened) by user programs, utilities, or other command input files.

Note

The CLOSE -ALL command should not be used in a command input file, as it closes all files, including the command input file from which this command is read. The message "End of file. Cominput. (Input from terminal.)" will be printed and input control will be switched to the terminal.

THE COMOUTPUT COMMAND

The COMOUTPUT command writes, into a specified file, both the output stream directed to the terminal by PRIMOS and the input presented to PRIMOS. The input may originate as direct typing, or come from a command file running under COMINPUT, PHANTOM or Batch. The resulting output file is a permanent record of the entire dialog.

Output to the terminal can be suppressed. Print suppression increases speed since it normally takes more time to write to a terminal than to a disk file.

The command format is:

COMOUTPUT [output-file] [options]

output-file is the pathname of the file to which the output stream is sent. options specify terminal and file output and control flow as described below.

Terminal Options

These can be used when the output file is first opened, or at any time before the command output file is closed. User input is always echoed at the terminal even if the -NTTY option is used.

<u>-NTTY</u>	Turn off terminal output.
<u>-TTY</u>	Turn on terminal output (default).



Error messages are printed in the output file and at the terminal, regardless of the terminal option selected. Output which is sent to the terminal from other users, such as messages from the supervisor terminal, is printed at the terminal but not in the output file.

### File Options

These stop or restart output to the command file. They may also be used to append output to an existing file.

- PAUSE      Stop output to command file; leave file open.
- CONTINUE    Resume output (halted by -PAUSE) to the command output file. Or, if at PRIMOS level, re-open an existing COMOUTPUT file and position the pointer so that new output will be appended.
- END        Stop output to command file; close file.

A BREAK turns terminal output on, but does not close the file. A LOGOUT turns terminal output on and also closes the command output file, as well as any other files the user has currently open. For example:

```
COMO FINTEST.COMO
```

opens the file FINTEST.COMO for output and positions the pointer to the start of the file. If FINTEST.COMO already exists, its previous contents will be deleted immediately. To open an existing file for appending, type:

```
COMO FINTEST.COMO -C
```

This opens the file FINTEST.COMO and positions the pointer at the end of the file.

### Closing Command Output Files

Command output files are normally closed by the COMO -END command. For example:

```
COMO TEST.COMO
SLIST RECORDS
COMO -END
```

Command output files may also be closed with the PRIMOS command.

```
CLOSE pathname
```



Substituting this command in the last example, we get:

```

COMO TEST.COMO
SLIST RECORDS
CLOSE TEST.COMO

```

### USING DATE AND TIME IN COMMAND FILES

#### The DATE Command

The command DATE prints the system date and time at the user terminal.

```

OK, DATE
11 Feb 82 14:53:28 Thursday
OK,

```

This feature allows command output files to be stamped with date/time information for identification, as an aid to program development and debugging. For example, the sequence of commands:

```

COMO TEST1.COMO
DATE
.
.
.
DATE
COMO -END

```

creates a file, TEST1.COMO. The first line of this file is the DATE command; the next line is the time and date of this interactive session.

DATE may also be included in command input files or in command files for Batch execution.

#### The TIME Command

The command TIME (abbreviation: T) entered at the user terminal prints the current values in the time accounting registers. These are: connect time, compute time, and disk I/O time.

```

OK, TIME
Time used: 01h 02m connect, 03m 05s CPU, 02m 05s I/O
OK,

```



Connect time is the time since LOGIN (in hours and minutes). CPU time is the time accumulated executing commands or using programs (in minutes and seconds). This does not include disk I/O time. Disk I/O time (in minutes and seconds) is the accumulated time for disk input and output. Disk I/O includes paging I/O time generated on the user's behalf. All times include system supervisor overhead caused by user requirements.

The TIME command can be given before and after executing a program. The time differences can be used to benchmark the program and measure efficiency as the program is optimized.

Example: The command input file BENCH07.COMI contains the following:

```

COMO BENCH07.COMO
/* TIMING TEST OF BENCH07 PROGRAM
DATE
/* GET START TIME VALUES
TIME
SEG TEST
/* GET STOP TIME VALUES
TIME
COMO -END
CO -TTY

```

The command CO BENCH07.COMI executes this command file. Upon completion, the output file BENCH07.COMO contains the following:

```

OK, /* TIMING TEST OF BENCH07 PROGRAM
OK, DATE
12 Feb 82 17:34:04 Friday
OK, /* GET START TIME VALUES
OK, TIME
Time used: 00h 03m connect, 00m 01s CPU, 00m 00s I/O.
OK, SEG TEST
The answer is 70
OK, /* GET STOP TIME VALUES
OK, TIME
Time used: 00h 03m connect, 00m 02s CPU, 00m 01s I/O.
OK, COMO -END

```

#### The RDY -LONG Command

The RDY -LONG command, discussed in Chapter 17, provides another way to measure program efficiency. After this command is given, each OK prompt includes the time of day, the amount of CPU time (in seconds) and the amount of I/O time (also in seconds) used since the last prompt.

```

OK, RDY -LONG
OK 09:21:29 0.284 0.324

```



The command RDY -BRIEF returns prompts to the normal form. This command is also illustrated in Chapter 17.

The following command file, BENCH07A.COMI, shows the use of the RDY -LONG command.

```

COMO BENCH07A.COMO
/*TIMING TEST OF BENCH07 PROGRAM
DATE
/*use RDY-LONG to show time between prompts
RDY -LONG
SEG TEST
COMO -END
CO -TTY

```

This program creates the following output file, BENCH07A.COMO

```

OK, /* TIMING TEST OF BENCH07 PROGRAM
OK, DATE
16 Feb 82 10:04:24 Tuesday
OK, /*use RDY -LONG to show time between prompts
OK, RDY -LONG
OK 10:04:27  2.078  2.645
SEG TEST
The answer is 70
OK 10:04:30  0.557  1.766
COMO -END

```

### PHANTOM USERS

The phantom user feature allows command file processing without tying up a terminal. Once a phantom process has been initiated, it is treated by PRIMOS as a separate process that is not associated with a terminal. The terminal is then made available for other uses.

The command file or CPL program run by the phantom process specifies the commands and their sequence, program invocations and necessary input data required to complete a particular job. Phantoms are used for long compilations, loadings, and executions that are debugged and require no interactive terminal input. Certain PRIMOS system utilities (for example, FAM, SPOOL) are implemented as phantom processes.

### Using Phantoms

A phantom user process is initiated by the command:

```

PHANTOM pathname [CPL-args ]
                  [file-unit]

```

pathname is the name of a CPL program or command input file.



If a COMINPUT file or special Phantom command file is to be run, then file-unit may be the PRIMOS file unit number on which the command file is to be opened. If omitted, file unit 6 is used. (File units may not be specified for CPL programs, which allocate their file units automatically.)

If a CPL program is being run as a phantom, then CPL-args are the arguments to be passed to the CPL program.

The PHANTOM command checks for available phantom processes. The number varies with each installation. The message:

No phantoms are available. FILENAME

is returned if no processes are available. Control is then returned to PRIMOS. When a phantom process is available, the message:

PHANTOM is user usernumber

is returned and the phantom user is logged in (under the same login-name as the invoker). usernumber is the number assigned by PRIMOS to the phantom process. Control returns to PRIMOS, the terminal is freed for other use, and the phantom command file is opened on the specified (or default) unit. PRIMOS then reads all further commands for the phantom user from the command file.

### Phantom Operation

Phantom processes should not execute programs which require input from an actual terminal. Such an instruction will abort and log out the phantom process.

While a phantom process is in operation, terminal output is suppressed unless a command output file has been opened by a COMOUTPUT command in the phantom command file. Output is then written to the COMOUTPUT file.

It is possible to initiate another phantom from a running phantom, in a manner similar to chained COMINPUT files. However, there is no guarantee that a phantom user process will be available when the process is requested by a command file.

With one exception, the final command in the last executed phantom command file should be LOGOUT. If it is not, the phantom will report an abnormal termination when it is logged out. The exception is a CPL phantom, which does not need the LOGOUT command for a normal logout.



Phantom Logout

At the completion of a job process, phantom users are automatically logged out. To cancel a phantom user process before completion, use the command:

LOGOUT -usernumber

usernumber is the PRIMOS-assigned phantom user number.

Any phantom can be logged out from the supervisor terminal. From a user terminal, a phantom can be logged out only if the terminal has the same user id as that which initiated the phantom.

Logout Notification

When a phantom logs out, notification is ordinarily sent to the terminal of the user who started the phantom. Normal logout is shown by a message such as:

Phantom 87: Normal logout at 11:27  
Time used: 00h 00m connect, 00m 04s CPU, 00m 05s I/O

Forced logout (the result of an error that halted the phantom program, a deliberate LOGOUT command, or the absence of LOGOUT as the final command in the phantom's final command file) results in a message such as:

Phantom 113: Abnormal logout at 15:49  
Time used: 00h 00m connect, 00m 04s CPU, 00m 03s I/O

In these messages, the figures following the phrase "time used" indicate elapsed time, CPU time, and I/O time used by the phantom process.

Controlling Logout Notifications: You can use the LON command to control whether the normal phantom logout notifications are sent to your terminal. The format is:

LON [-OFF]  
[-ON ●]

The command LON -OFF prevents phantom logout notifications from printing at your terminal. You may wish to use LON -OFF when you do not wish to receive messages from PRIMOS, as, for example, when you are using the Editor or running a COMOUTPUT file.

The command LON by itself or LON -ON allows you to receive phantom logout notifications, including any messages waiting because of a previous LON -OFF state. LON -ON is the default state.



If the user who started the phantom logs out before the phantom completes its job, logout notification cannot be sent to the user's terminal. It is possible, however, for users to set up programs to record phantom logout notifications. This is done using the subroutines LON\$R and LON\$CN. For information on these subroutines, see the Subroutines Reference Guide.

### Phantom STATUS Information

The STATUS USERS command (summarized in Appendix G) provides a list of all the users in the system, their login numbers, assigned line numbers, and so on. Phantom users are distinguished by the code "phant" in the line number field of a STATUS list. For example:

OK, STATUS USERS

User	No	Line	Devices
SYSTEM	1	asr	<PRODUC> SMLC00 SMLC01 AL077
SURE	5	3	SYSZ<FIRST> <PRODUC>
PXO	7	5	<SALESR> <PRODUC>
NEDDIE	8	6	<PRODUC>
RUNCK	64	rem	<PRODUC> (from SYSX )
STEVE	65	rem	<PRODUC> (from SYSX )
JONNIE	66	rem	<PRODUC> (from SYSX )
CROWN	68	rem	<PRODUC> (from SYSX )
TONYB	69	rem	<SALESR> (from SYSY )
NETMAN	88	nsp	<PRODUC>
DOUGLAS	92	slave	<PRODUC>
SLAVE\$	93	slave	<PRODUC>
TOMMYQ	94	slave	<PRODUC>
SLAVE\$	97	slave	
BATCH_SERVICE	101	phant	<SPOOL> (2)
SYSTEM	103	phant	<SALESR>
YTSMAN	104	phant	<PRODUC>
FTSSRV	105	phant	<PRODUC>
LM.B	106	phant	<PRODUC>
SYSTEM	107	phant	SYSA<ORSYSA> <SPOOL> <PRODUC> PRO
SYSTEM	108	phant	SYSB<ORSYSB> <SPOOL> <PRODUC> PRI

OK,

The STATUS command is fully discussed in PRIMOS Commands Reference Guide.



Example of Phantom Command File

The phantom command file TEST.PH contains the following commands:

```
/*BEGIN TEST OF PHANTOM
COMOUTPUT TESTPH.COMO
DATE
/*COMPILE THE PROGRAM IN 64V MODE
FIN TEST -64V
/*LOAD THE PROGRAM
SEG -LOAD
LO TEST
LI
SA
MAP LOADTEST.MAP 7
MAP UNSATISFIED.MAP 3
QU
/*PHANTOM TEST COMPLETED
DATE
/*COMO -E would normally go here.
/* It has been omitted so the logout sequence
/* could be shown in the comoutput file.
LOGOUT
```

When a phantom is invoked at the terminal by PH TEST.PH, the terminal interactive dialog is:

```
OK, PH TEST.PH
PHANTOM is user 61
OK,
```

The contents of the command file, TESTPH.COMO, created by the phantom are:

```
OK, DATE
16 Feb 82 16:54:32 Tuesday
OK, /*COMPILE THE PROGRAM IN 64V MODE
OK, FIN TEST -64V
0000 ERRORS [<.MAIN.>FIN-REV19.0]
OK, /*LOAD THE PROGRAM
OK, SEG -LOAD
[SEG rev 19.0]
$ LO TEST
$ LI
LOAD COMPLETE
$ SA
$ MAP LOADTEST.MAP 7
$ MAP UNSATISFIED.MAP 3
$ QU
OK, /*PHANTOM TEST COMPLETED
OK, DATE
16 Feb 82 16:54:48 Tuesday
OK, /*COMO -E would normally go here.
OK, /* It has been omitted so the logout sequence
```



OK, /\* could be shown in the comoutput file  
OK, LOGOUT

GRACE (user 118) logged out Tuesday, 16 Feb 82 16:54:48.  
Time used: 00h 00m connect, 00m 04s CPU, 00m 05s I/O.



ALL INFORMATION CONTAINED HEREIN IS UNCLASSIFIED

DATE 01-11-2001 BY 60322 UCBAW/STP



# 11

## Batch Job Processing

### INTRODUCTION

Batch is the most flexible of the PRIMOS job processing utilities. Any CPL program or command file that will run under PRIMOS can be run as a Batch job. This means that users may write CPL programs for submission as Batch jobs without including special Batch commands. Users may also run existing COMINPUT and PHANTOM files as Batch jobs; Batch will accept them all.

Batch offers flexibility in job scheduling and execution control. Each Batch queue runs users' jobs as a phantom. These phantoms run "in the background" of the system: that is, they run concurrently with interactive jobs, but at somewhat lower priorities. Thus, they use only small amounts of CPU time when interactive use is heavy, but utilize large amounts of CPU time when interactive use is light or absent. Furthermore, Batch jobs may be held in their queues by operators, then released to run at appropriate times. Thus, extremely long jobs, such as file updates and backups, can be set up as Batch jobs during the day, then run under operator control at night.

Each Batch queue is a separate entity, defined by the System Administrator to be particularly hospitable to certain types of jobs. Queues designed for short jobs have a fairly high schedule priority, but a short timeslice; queues designed for normal jobs have slightly lower priorities and normal timeslices. Queues designed for long jobs have low priorities but large timeslices. The queues for short jobs will thus run fastest, as they can operate during times of heavier interactive use. The other queues will take more advantage of periods of lighter activity. By using the BATGEN (BATch GENeration) command,



explained below, users can see what queues are available and what their characteristics are. They can then submit their jobs to the appropriate queues.

### USING THE BATCH SUBSYSTEM

Users communicate with the Batch subsystem through four commands: BATCH, BATGEN, JOB, and \$\$ JOB. With these commands, they can:

- Submit jobs (JOB)
- Set job parameters (JOB, \$\$ JOB)
- Modify, cancel, abort, or restart jobs (JOB)
- Monitor subsystem usage (BATCH)
- Monitor queue characteristics and availability (BATGEN)

These operations are described in this chapter.

### SUBMITTING BATCH JOBS

To submit a job, use the command:

```

JOB pathnamel [
  -ACCOUNTING info
  -ARGS cpl-arguments
  -CPL
  -CPTIME { seconds
           NONE
           minutes
           NONE
         }
  -ETIME {
  -FUNIT number
  -HOME pathname2
  -PRIORITY value
  -QUEUE queue name
  -RESTART { YES
            NO
          }
]

```

pathnamel is the pathname of the file to be submitted to Batch.

Batch will then send a "job submitted" response announcing the job's job-id number and reminding the user of the home directory for the job (if he didn't use the -HOME option).



For example:

```
OK, job TWIG
[JOB rev 19.0]
Your job, #00015, was submitted to queue Normal-1.
Home=<FOREST>BEECH>BRANCH4
OK,
```

As this example shows, jobs may be submitted without options. The Batch monitor places these jobs in the first available queue and uses that queue's default values for all necessary parameters. On the other hand, users may specify queue and/or parameters, using the JOB command's options as described in the list that follows.

### Notes

All numbers must be decimal integers.

If a job is submitted from a passworded directory (including your current attach point), the -HOME option, explained in the list, must be used. The password must be included in the pathname, and the pathname must be enclosed in single quotation marks; for example: JOB filename -HOME 'dir-name password'.

The options that you can use to control your Batch jobs are listed below.

<u>Option</u>	<u>Description</u>
{ -ACCOUNTING } info { -ACCT }	Allows the user to specify accounting information for his job. The information must be 80 characters or fewer in length. It can not be an explicit register setting (octal number) or be preceded by an unquoted minus sign. If the information contains spaces, commas, or comment delimiters (/*), it should be enclosed in apostrophes. For example: <u>-ACCT 'OK, HERE WE GO'</u> . The information will be included in job displays, but will not be used in running the job.
-ARGS cpl-args	Used to pass CPL arguments to the job being processed. -ARGS must be the last option issued on a command line because everything that follows the -ARGS option on the command line (except comments) is assumed to be the CPL arguments being passed. JOB doesn't read the CPL arguments; it just passes them to the CPL file when execution of the file begins.



- CPL** Runs submitted file as a CPL file, no matter what the file's name is. (You do not need to use the **-CPL** option if the **-ARGS** option is used or if the filename ends in **.CPL**.)
- CPTIME** { seconds }  
                  { NONE } Specifies the maximum amount of CPU time (in seconds) to be allotted to the job. **NONE** requests that no time limit be placed on the job. If the job exceeds the time limit, it will be aborted.
- ETIME** { minutes }  
                  { NONE } Specifies (in minutes) the elapsed time to be allowed before the job is aborted. Details are the same as those for **-CPTIME**.
- FUNIT** number Specifies the file unit to be used for command input. Permissible values range from 1 to 126, unless the System Administrator has set a lower limit. The smallest possible range is 1 to 16. The default value depends upon the queue to which the job is submitted. It is usually 6.
- FUNIT** may not be used in CPL jobs, as CPL jobs receive dynamically assigned file units. Attempts to use **-FUNIT** for CPL jobs result in the following message:
- Illegal combination. -FUNIT (JOB)
- HOME** pathname2 Specifies the UFD in which a job is to run. Using this option has the same effect as providing an **ATTACH** command as the first line of the command file. The pathname for a **-HOME** option, however, may not be a null specification or a relative pathname (that is, it may not begin with **\*>**), and may not exceed 80 characters in length.
- PRIORITY** value Determines the job's priority within its queue. Possible values are from 0 to 9, with 9 being the highest (most favored) priority. The default depends on the queue.
- QUEUE** queueName Names the queue in which the job should be placed. (To learn the names and characteristics of queues, use the **BATGEN -DISPLAY** command.) If this option is not specified, the Batch monitor places the job in the first unblocked queue where the job will fit.



{	-RESTART	}	{	YES	}	Determines whether a job can be restarted following an ABORT or a system shutdown. The default is always -RESTART YES.
{	-RST	}	{	NO	}	

The following example illustrates a job submitted with options:

```
JOB TWIG -ETIME NONE -PRIORITY 4 -ARGS TEST
```

If, for any reason, the Batch monitor cannot accept the job as submitted, it will send the user error messages containing the information he needs to resubmit the job successfully. These messages are listed in Appendix D; they are generally self-explanatory.

#### Note

The operator, but not the user, has the power to hold a user-submitted job indefinitely in a Batch queue. Execution of a held job is delayed until the operator releases the job.

#### SUPPLYING OPTIONS VIA THE \$\$ JOB COMMAND

Any or all of the JOB command's options may be given in the first non-comment line of the command file itself by the command:

```
$$ JOB { * } {user-id} {options}
```

If a specific user-id is given on the \$\$ JOB command line, only a user logged in with that user-id can submit the file. If an asterisk (\*) is used instead, any user can submit the file.

The following example illustrates the use of \$\$ JOB:

```
$$ JOB JONES -HOME JONES>REPORTS -CPTIME 30
```

Users will probably find the \$\$ JOB command most helpful for parameters that they expect to remain constant whenever the job is submitted. The JOB command options will probably be most helpful for parameters that change from submission to submission.

This is true because parameters given on the JOB command line at Primos level override parameters given on the \$\$ JOB command line within your file. For instance, if you specified "\$\$ JOB \* -CPTIME NONE" in your file, but wanted to run the job in a queue which had a CPU time limit, the command "JOB TEST SCORES -CPTIME 180 -QUEUE FAST" would run the job in queue FAST with a CPU time limit of 180 seconds.



Note

With one exception, any Batch command file, even one including a \$\$ JOB command, can be run interactively. The exception is a file using the \$\$ JOB -HOME option. When run interactively, the \$\$ JOB line will be ignored, and no ATTACH will be done. In this case, add an ATTACH command to the file immediately following the \$\$ JOB line.

Both the JOB and \$\$ JOB command lines may be up to 160 characters in length. If a command line exceeds the physical length of the terminal line, continue input without striking the return key.

CONTROLLING BATCH JOBSModifying Parameters

To modify a job's parameters after it has been submitted, use the -CHANGE option of the JOB command:

JOB { jobname } {	{	{	-CHANGE	}	{	-ACCOUNTING	info	}
			{			-ARGS	cpl-args	
						-CPL		
						-OPTIME	{ seconds }	
							NONE	
						-ETIME	{ minutes }	
							NONE	
						-FUNIT	number	
						-HOME	pathname	
						-RESTART	{ YES }	
	NO							

jobname and job-id identify the job as follows:

ArgumentDescription

jobname

The name of the file being run. If the job was submitted as a pathname (e.g., JOB FELLOWSHIP>HOBBITS>FRODO), its jobname is the final element of the pathname (e.g., FRODO). More than one active job may have the same jobname if the same file has been submitted more than once as a Batch job.

job-id

A 5-digit number assigned to a job by the monitor when the job is placed in a queue. Each active job has a unique job-id number.



The following two examples illustrate the use of the -CHANGE option:

```
JOB #10039 -CHANGE -ACCT 'new research' -HOME ECON>STATS
JOB TEST_SCORES -CHANGE -FUNIT 8 -RESTART YES
```

A job's -QUEUE and -PRIORITY options cannot be changed. If they are in error, the job must be cancelled and resubmitted.

### Restarting Jobs

To restart a job already running, use:

```
JOB { jobname }
   { job-id } -RESTART
```

You would normally wish to use the -RESTART option in conjunction with the JOB -CHANGE command, when you need to change an executing job. The JOB -RESTART command causes a job to abort and later to restart if it is in restartable state. If the job cannot be restarted, it will abort only. Note that this option works only on executing jobs.

The following example illustrates how to change a job that is already running:

```
JOB SCORES -CHANGE -HOME RESRCH>STATS>NEWSTATS
JOB SCORES -RESTART
```

The JOB -CHANGE option marks the changes in the job's status; the JOB -RESTART command terminates execution, and then flags the job as ready for restarting under its new conditions.

### Note

Distinguish between the -RESTART YES/NO option and the -RESTART command. The option always takes an argument; it signals whether or not a job may be restarted. The -RESTART command takes no argument; it aborts and attempts to restart the job.

### Cancelling Jobs

To prevent a waiting or held job from running, use the command:

```
JOB { jobname }
   { job-id } -CANCEL
```

This command will not halt a job that is already running, but it will mark that job as not restartable.



Aborting Jobs

To terminate execution of a job already running, use:

```
JOB {jobname}
    {job-id} -ABORT
```

This command cancels a waiting or held job and forces a running job to log itself out immediately.

Tips on Using Control Options

The JOB -CHANGE, -CANCEL, -ABORT, and -RESTART commands will accept a jobname in place of a job-id only if the user has only one active job of that name. Thus, if file TEST has been submitted once, the command "JOB TEST -CANCEL" will work. But if two submissions of TEST (for example, #10057 and #10064) are active, you will receive the following message:

Multiple jobs with this name (use internal name).

In this case, you must use the job-id (or "internal name") to tell the Batch system which job to cancel.

Only one of the Batch control options (-CHANGE, -CANCEL, -ABORT, and -RESTART) may appear on the same JOB command line. For example, JOB TEST -ABORT -RESTART is illegal.

MONITORING BATCHMonitoring the Progress of Your Own Jobs

You may request information on the progress of your own jobs within the Batch system by specifying:

```
JOB {jobname} { -STATUS }
    {job-id}  { -DISPLAY }
```

The -STATUS and -DISPLAY options govern the amount of information to be shown. The jobname and job-id arguments allow you to specify the jobs on which you want information, as follows:

<u>Argument</u>	<u>Description</u>
jobname	Use the <u>jobname</u> to request information on all active jobs with this name (useful with multiple submissions of a file).
job-id	Use the <u>job-id</u> number to request information on one job only.



Omitting the jobname and job-id arguments requests information on all the user's active jobs.

<u>Option</u>	<u>Description</u>
-STATUS	Prints out the job's <u>jobname</u> and <u>job-id</u> , the name of the queue in which it is placed, and its execution status: whether it is held, waiting, executing, completed, or aborted.
-DISPLAY	Provides detailed information on the job. This includes the status information above, plus values for all JOB and \$\$ JOB command options -- both those specified by the user and those assigned from queue-defined defaults. -DISPLAY also prints the home UFD of the job, including any directory passwords in the pathname.

Example of the JOB -STATUS Command: Assume that user SUSAN has submitted a job named EXAMP. The JOB -STATUS command generates the following report:

OK, JOB -STATUS  
[JOB rev 19.0]

Job status listing for user SUSAN:

Jobid#	State	External name	Queue
#00015	executing	examp	Normal-1

Example of the JOB -DISPLAY Command: Assume, as above, that user SUSAN has submitted a job named EXAMP. The JOB -DISPLAY command generates the following report:

OK, JOB -DISPLAY  
[JOB rev 19.0]

Job examp(#00015), user SUSAN executing (queue Normal-1).  
Submitted today at 3:46:03 p.m., initiated today at 3:46:05 p.m.  
Funit=6, priority=5, cpu limit=None, elapsed limit=None.  
Home ufd=<DISK>SUSAN

If a job is restarted, the following message appears after the "Submitted..." line in the report above:

(This job has already executed n times).

n is number of previous executions (or the number of restarts).



Using the MESSAGE Command

Another way to monitor your Batch jobs is to have the jobs send messages to your terminal announcing the completion of key portions of the job. To do this, use the MESSAGE command (explained in Chapter 17), as shown below.

Messages from CPL Programs: CPL programs put their messages in &DATA groups. The format is:

```
&DATA MESSAGE { -usernumber }
               { user-id   }
               text of message
&END
```

For example:

```
&DATA MESSAGE BEECH
      Customer list update completed
&END
```

Messages from COMINPUT Files: Command input files write the text of their messages as comment lines:

```
MESSAGE { -usernumber }
        { user-id   }
/* text of message
```

For example:

```
MESSAGE BEECH
/* Customer update completed
```

Using this format prevents errors from occurring if the recipient of the message is not logged in at the time the message is to be sent.

Monitoring Usage of the Entire Batch Subsystem

You may request information on usage of the entire Batch subsystem with the command:

```
BATCH { -STATUS }
      { -DISPLAY }
```



The BATCH -STATUS command prints a one-line summary giving the number of waiting and held jobs, the number of queues that have waiting and held jobs, and the number of executing jobs. If there are waiting and held jobs as well as executing jobs, the total number of active batch jobs will also be printed. For example:

OK, BATCH -STATUS  
[BATCH rev 19.0]

6 batch jobs; 4 waiting or held jobs in 2 queues; 2 executing jobs.

The BATCH -DISPLAY command prints information on Batch usage in two tables. The first table prints the number of jobs waiting or held in each queue. The second table lists the number of jobs currently executing and identifies each by user-id, job-id number, phantom usernumber, and queue. For example:

OK, BATCH -DISPLAY  
[BATCH rev 19.0]

Number of waiting and held jobs:

Queue	Jobs
Normal-1	1
Normal-2	3

Total= 4 (2 queues)

2 currently running jobs:

User	Jobid#	#	Queue
MAPLE	#10032	114	Normal-2
OAK	#00172	117	Normal-1

### Monitoring Characteristics and Availability of Queues

You may request information on Batch queues by specifying:

BATGEN { -STATUS  
          -DISPLAY [queuenam] }



The BATGEN -STATUS command lists the currently defined queues and notes whether each is blocked (not accepting jobs) or unblocked (available for use). For example:

```
OK, BATGEN -STATUS
[BATGEN rev 19.0]
```

Queue:	Status:
Express	unblocked
Normal-1	unblocked
Normal-2	blocked

The BATGEN -DISPLAY [queuename] command identifies and gives full characteristics of the queuename specified. If queuename is omitted, BATGEN -DISPLAY gives the characteristics for all queues. For example:

```
OK, BATGEN -DISPLAY NORMAL
[BATGEN rev 19.0]
```

```
Queue name = NORMAL, unblocked.
Default cptime=30, etime=None, priority=5;
Maximum cptime=180, etime=None; Funit=6;
Delta rlevel=1; Timeslice=20;
```

In this example, NORMAL is the queue's name. Unblocked means that the queue is accepting jobs for queueing and execution. The default cptime and etime values will apply to jobs that don't specify their own CPU time or elapsed time options. The maximum cptime and etime values are the largest allowed for any job running from the queue. Priority and funit are default values for those options.

Delta rlevel and timeslice refer to run-time priorities. Queues with high delta rlevels and large timeslices are best for long jobs; queues with low delta rlevels and short timeslices are best for short jobs. The queue in the example is designed for average jobs.



**PART III**  
**System Facilities**



PART III

System Facilities



# 12

## File-Handling Utilities

### INTRODUCTION

This chapter introduces you to Prime's basic file-handling utilities. These utilities allow you to:

- Sort one or more unsorted files into one sorted file (SORT)
- Merge several sorted files into one sorted file (SORT)
- Compare files with each other (CMPF)
- Resolve differences between files (MRGF)
- Join text files sequentially (CONCAT)

### SORTING FILES (SORT)

The SORT command sorts up to 20 files, on up to 50 keys, into a single output file. SORT preserves the order of input for records with equal keys (that is, it is a stable sort).

Most sorts are done on ASCII files (also called compressed files), such as those created by the text editor (ED). The following discussion emphasizes how to do ASCII sorts. In addition, SORT can process uncompressed files, variable-length files (also called binary files), and fixed-length files. The basic format for using SORT is the same for every file type, but details vary from type to type. The PRIMOS



Commands Reference Guide contains complete information and sorting instructions for each file type.

SORT can also sort files using the EBCDIC collating sequence. For details, see the PRIMOS Commands Reference Guide.

### Using SORT

To use SORT, provide information in a three-step or four-step sequence, as follows:

1. Give the SORT command (and any desired options).
2. Specify the sort files and number of sort fields, either by a simple parameter list or by the use of keywords.
3. Specify the starting and ending columns of sort fields (keys).
4. If -MERGE is specified, enter additional filenames.

SORT normally specifies the information it wants at steps 2, 3, and 4. However, once you are familiar with the prompt dialog, you can suppress the printout by using the -BRIEF option with the command line. If -BRIEF is specified, simply give the information line by line in the same order SORT asks for it. Refer to the sample sort contained in the section A Mergesort Example for an example of the SORT dialog.

### The SORT Command

To invoke SORT, give the SORT command, either by itself or accompanied by one to four options:

```
SORT [-BRIEF] [-SPACE] [-MERGE] [-TAG •]
      [-NONTAG]
```

SORT's options are as follows:

<u>Option</u>	<u>Meaning</u>
-BRIEF	SORT program messages and prompts are not printed at the users terminal.
-SPACE	Any blank lines in the input file(s) are deleted from the SORT output file.
-MERGE	A merge of presorted files is requested.



- TAG        A TAG sort (described below) is requested.
- NONTAG     A NONTAG sort (described below) is requested.

A TAG sort is specified when large files are sorted. For unordered files, it is a faster sort than NONTAG. Internally, the TAG sort stores input records separate from the key data. After all keys have been sorted and merged, the corresponding records are then located and output.

A NONTAG sort may be specified for smaller or well-ordered input files. Internally, the NONTAG sort stores each input record with its sort key in the work file. This eliminates the search for each record after merging, but requires more disk space.

#### Note

Output files may be a different type than input files.

SORT responds by requesting:

- The name of the file to be sorted
- The name of the output file to be created
- The number of keys for the sort (default is 1)

#### Simple File and Key Specifications

The simplest type of sort reads one unsorted ASCII file and creates another sorted ASCII file. To specify this sort, simply list the filenames and number of keys (if greater than 1) on one line, then list the starting and ending columns for each key field on a separate line. If the data within a key field are to be sorted by some code other than straight ASCII, type a space and the data type after the ending column. (The SORT dialog will list data types and their codes. They are also explained, in greater detail, in the PRIMOS Commands Reference Guide.) If the sort on any key is to be done in reverse (descending) order, type a space and an "R" after the ending column or data type. For example, to sort a list of names and addresses in ascending order, the entire entry of 80 characters might constitute the sort field, and the commands would run:

```
OK, SORT -BRIEF
JUMBLED.NAMES  NEAT.NAMES
1      80
```

Unless the -MERGE option has been specified, sorting begins when the last pair of column numbers is entered. When the sort is complete,



SORT prints at the terminal the number of passes needed for the sort and the number of items (i.e., lines) placed in the output file, and then returns to PRIMOS.

### Other File Specifications

If you are sorting more than one file, give all filenames plus the number of keys on a single line in the following format:

```
-INPUT inputfile [...-INPUT inputfile] -OUTPUT outputfile -KEYS n
```

For example:

```
OK, SORT -BRIEF
-INPUT CHAOS.1 -INPUT CHAOS.2 -OUTPUT ORDER -KEYS 2
1 10
15 20 R
```

BEGINNING SORT

```
PASSES      2      ITEMS      10
```

[SORT-REV18.0]

OK,

If you are sorting uncompressed or fixed length files, or if you are sorting binary files using ASCII keys, you will have to specify additional file information (via keywords) along with the filenames. See the PRIMOS Commands Reference Guide for details.

### Key Specifications

SORT recognizes 13 types of keys. ASCII files (compressed and uncompressed) can use seven of them: A and AU for alphanumeric data, U, LS, TS, LE and TE for numeric data. (The other six keys are discussed in PRIMOS Commands Reference Guide.)

Alphanumeric keys: The two alphanumeric keys are ASCII (A), which sorts in a strict ASCII sequence, and ASCII, upper and lower (AU), which sorts all alphanumeric characters as if they were uppercase. (The ASCII sequence is given in Appendix C.) The default key type is strict ASCII (A).



Given the four words, APPLE, alphabet, WHY, and whynot, ASCII (A) produces:

```
APPLE
WHY
alphabet
whynot
```

AU produces:

```
alphabet
APPLE
WHY
whynot
```

Numeric keys: Three common numeric keys for ASCII sorts are:

- U Numbers without plus or minus signs
- LS Numbers preceded by plus or minus signs  
(Numbers without signs are considered positive.)
- TS Numbers followed by plus or minus signs  
(Numbers without signs are considered positive.)

(The LE and TE keys, which have the sign embedded in the numeral, are explained in the PRIMOS Commands Reference Guide.)

Here is an example of a sort on an LS key:

```
OK, SORT -BRIEF
NUMBERS NUMBERS.1
1 10 LS
```

BEGINNING SORT

```
PASSES      2      ITEMS      7
```

[SORT-REV18.0]

```
OK, SLIST NUMBERS.1
-9999
-8205
-6783
4114
+5483
8265
+9765
```

OK,



Additional Filenames for the -MERGE Option

After key fields have been specified using the -MERGE option, SORT asks for the number of additional files to be merged. If you have already listed all input files with the -INPUT format, this number is 0. Otherwise, give the number of additional files and then the names of the files, one name per line. When the last name is entered, the mergesort begins. When the merge is complete, SORT prints the number of passes and returns to PRIMOS.

A Mergesort Example

Here is an example of a mergesort. Assume we have created two transaction files, in which each line (record) has the following format: a transaction number in columns 1-5, a credit or debit notation in column 6, a customer name in columns 8-17, a customer ID number in columns 19-25, and other data in the remaining columns. Each file has been sorted by customer name, customer ID, and transaction number (in reverse order, so that most recent transactions come first). Now we are going to merge the two files, sorting on the same three keys. The sort, with the full SORT dialog, is as follows:

OK, SORT -MERGE

SORT PROGRAM PARAMETERS ARE:

INPUT TREE NAME — OUTPUT TREE NAME FOLLOWED BY  
NUMBER OF PAIRS OF STARTING AND ENDING COLUMNS.

CUST.CREDITS CUST.ACCTS 3

INPUT PAIRS OF STARTING AND ENDING COLUMNS  
ONE PAIR PER LINE—SEPARATED BY A SPACE.

FOR REVERSE SORTING ENTER "R" AFTER DESIRED  
ENDING COLUMN—SEPARATED BY A SPACE.

FOR A SPECIFIC DATA TYPE ENTER THE PROPER CODE  
AT THE END OF THE LINE—SEPARATED BY A SPACE.

"A" - ASCII

"I" - SINGLE PRECISION INTEGER

"F" - SINGLE PRECISION REAL

"D" - DOUBLE PRECISION REAL

"J" - DOUBLE PRECISION INTEGER

"U" - NUMERIC ASCII, UNSIGNED

"LS" - NUMERIC ASCII, LEADING SEPARATE SIGN

"TS" - NUMERIC ASCII, TRAILING SEPARATE SIGN

"LE" - NUMERIC ASCII, LEADING EMBEDDED SIGN

"TE" - NUMERIC ASCII, TRAILING EMBEDDED SIGN

"PD" - PACKED DECIMAL

"AU" - ASCII, UPPER LOWER CASE SORT EQUAL

"UI" - UNSIGNED INTEGER



DEFAULT IS ASCII.

8 17

19 25

1 5 R

INPUT THE NUMBER OF ADDITIONAL FILES TO BE MERGED. (MAX= 10): 1

INPUT FILES TO BE MERGED, ONLY ONE PER LINE.

CUST.DEBITS

BEGINNING MERGE

PASSES 1 ITEMS 10

[SORT-REV18.0]

OK, SLIST CUST.ACCTS

89424+ Jones	BR9438	other data about transaction
81884- Jones	BR9438	other data about transaction
12345+ Jones	BR9438	other data about transaction
67340- Jones	XL1489	other data about transaction
54936+ Jones	XL1489	other data about transaction
49480- Jones	XL1489	other data about transaction
86889+ Smith	CS4192	other data about transaction
29622+ Smith	CS4192	other data about transaction
23220- Smith	CS4192	other data about transaction
21220+ Smith	CS4192	other data about transaction

OK,

#### COMPARING FILES (CMPF)

The PRIMOS command CMPF permits the simultaneous comparison of up to five ASCII files of varying lengths. The format is:

CMPF file-1 file-2 [.....file-5] [options]

The first file, file-1, is treated as the original file against which the other files are compared. The CMPF command produces output indicating which lines have been added, changed, or deleted in the other files.



The options that may be specified are:

<u>Option</u>	<u>Function</u>
-BRIEF	Suppresses the printing of differing lines of text of files being compared. Only identification letters and line numbers are printed.
-MINL number	Sets the minimum number of lines that must match after a discrepancy between files is found. Needed in order to resynchronize file comparison. Default = 3 lines.
-REPORT filename	Produces a file with specified filename, containing the differences found between compared files (in lieu of displaying them at the terminal during the comparison process).

After a difference between the original file and another specified file has been discovered, CMPF attempts to resynchronize the files for comparison. This occurs only when a certain number of lines match in all the files being compared. The default value is 3, but can be changed in the -MINL option. The comparison process continues until another difference is found.

When line differences are reported, either at the terminal or in a report file, each line from the original file is indicated by the letter A, followed by the line number of the line containing discrepancies. The corresponding lines of other files are indicated in the same manner, using letters B through E respectively.

For example, consider the following two files:

FILEA

The  
quick  
brown  
fox  
jumps  
over  
the  
lazy  
dog

FILEB

The  
swift  
red  
fox  
jumps  
over  
the  
dog



A CMPF comparison of these two files works as follows:

OK, CMPF FILEA FILEB  
GO

A2            quick  
A3            brown  
CHANGED TO  
B2            swift  
B3            red

A8            lazy  
DELETED BEFORE  
B8            dog

COMPARISON FINISHED.  
2 DISCREPANCIES FOUND.

OK,

#### MERGING TEXT FILES (MRGF)

The MRGF command merges up to five ASCII files. The format is:

MRGF file-a [file-b ...file-e] -OUTF outfile [options]

The first file specified is treated as the original file, and it is assumed that changes have been made to this file to produce the other files. Pathnames may be used to specify files to be merged. Unchanged lines of text and nonconflicting changes between files are automatically copied to the output file, outfile. The original source files (file-a, file-b, and so on) are not changed; outfile is built from them.

When corresponding lines of text in the files differ, the user is asked by the MRGF program to solve the conflicts. This is done by entering an interactive mode in which the user can specify the contents of the output file. In this mode, the subcommand x causes all the queried lines from file-x to be inserted into outfile. x = A, B, C, D, or E; A signifies file-a, B signifies file-b, and so on. The subcommand xn causes line n from file-x to be inserted.

New text can be inserted by entering a blank line at the terminal (thus sending MRGF into input mode), typing the new text, and then typing another blank line. No text editing can be performed on lines thus input, and no expansion of tab characters will be done. The lines must be entered character-for-character as they are to appear.

The subcommand GO terminates editing and proceeds with the merge.



The options taken by the MRGF command are similar to those for the CMPF command. There is an additional option, **-FORCE**, which causes file-b to be the preferred file if conflicts exist between several files. No MRGF interactive dialog will be generated when conflicts arise if the **-FORCE** option is used. file-b is assumed "correct" and the other files forced to comply with it.

For example, consider the following two files:

<u>MAXIM</u>	<u>NOMAXIM</u>
A	The
rolling	rolling
stone	old
gathers	oaken
no	bucket
moss	holds
	no
	milk

A merging of these two files might proceed as follows:

OK, MRGF MAXIM NOMAXIM -OUTF MIX -MINL 1  
[MRGF 19.0]

A1            A  
CHANGED TO  
B1            The  
EDIT.  
B  
GO

A3            stone  
A4            gathers  
CHANGED TO  
B3            old  
B4            oaken  
B5            bucket  
B6            holds  
EDIT.  
B3  
A3  
B6  
(CR)  
INPUT.  
(it would seem)  
(CR)  
EDIT.  
GO



```

A6      moss
CHANGED TO
B8      milk
EDIT.
A
GO

```

```

MERGE FINISHED.
3 MANUAL CHANGES.
NO AUTOMATIC CHANGES.

```

```

OK,

```

The subcommand B selects NOMAXIM's version of the differing line in the first edit group and inserts it into MIX. The subcommand GO returns MRGF to the merge activity.

The subcommands B3, A3, and B6 insert these lines ("old", "stone", and "holds"), in this order, into MIX. An extra carriage return sends MRGF into input mode to accept the string "(it would seem)". A second extra carriage return sends MRGF back to edit mode. The subcommand GO again returns MRGF to the merge activity.

The subcommand A selects MAXIM's version of the differing line in the next edit group and inserts it into MIX. The subcommand GO once again returns MRGF to the merge activity.

The output file MIX now contains the following:

```

MIX
The
rolling
old
stone
holds
(it would seem)
no
moss

```

More detailed information on MRGF appears in the PRIMOS Commands Reference Guide.



JOINING SEVERAL FILES SEQUENTIALLY (CONCAT)

The CONCAT command concatenates files into a single file, which can then be printed via the SPOOL command. You might wish to use CONCAT if you are spooling many files and want:

- to save paper and printing time by reducing the number of header pages to be printed.
- to ensure that files are printed in a specific order. (Otherwise, SPOOL will print shorter files before longer ones.)

The format for CONCAT is:

CONCAT new-filename [-options]

Options govern the format of the print-out and the disposition of the files. For details, see the CONCAT discussion in the PRIMOS Commands Reference Guide.

When you give the CONCAT command without options, CONCAT goes into input mode. It asks for the names of the files to be concatenated, and prints a colon prompt. Type the filenames, one per line. A null line (carriage return) signals the end of list. CONCAT then goes into command mode, and prints a right-angle prompt. You type QUIT to end the session. (You can also type "INPUT" to return to input mode; or you can give various formatting commands, which are explained in the PRIMOS Commands Reference Guide.)

A sample session might be:

OK, CONCAT TRIPLET  
[CONCAT Rev 19.0]

Enter filenames, one per line:

: FIRST  
: SECOND  
: THIRD  
: (CR)

> Q

OK,



If the file TRIPLET already exists, CONCAT asks:

OK TO MODIFY OLD TRIPLET?

Answering NO returns you to PRIMOS command level. Answering YES prompts a second question:

OVERWRITE OR APPEND:

Answering OVERWRITE causes CONCAT to replace the old TRIPLET with a new one. Answering APPEND preserves the existing contents of TRIPLET and adds the new files at its end.



100-100000-100000

100-100000-100000

100-100000-100000

100-100000-100000

100-100000-100000



# 13

## Using Tapes and Cards

### ACCESSING DATA ON TAPES AND CARDS

Existing source programs stored on punched cards, magnetic tape, or punched paper tape can easily be read into disk files using PRIMOS-level utilities. In addition, the punched card and magnetic tape transfer utilities will translate BCD or EBCDIC representation into ASCII representation, an operation that saves time and effort. This chapter outlines how to use these utilities. Further information appears in the Magnetic Tape User's Guide and in the PRIMOS Commands Reference Guide.

Subroutines and other installation-dependent operations may be altered to conform to PRIMOS by using the Editor (ED) described in Chapter 4.

The general order of operations for input from a peripheral device is:

1. Obtain exclusive use of the device with the ASSIGN command.
2. Transfer programs with appropriate utility.
3. Relinquish exclusive use of the device with the UNASSIGN command.



Assigning a Device

Assigning a device gives the user exclusive control over that peripheral device. The PRIMOS-level ASSIGN command is given from the terminal:

ASSIGN device [-WAIT]

device is a mnemonic for the appropriate peripheral:

CR[n]	Card Reader <u>n</u> ( <u>n</u> = 0,1)
PTR	Paper Tape Reader
MTp <u>dn</u> [- <u>ALIAS</u> MTl <u>dn</u> ]	Magnetic Tape Unit <u>p<u>dn</u></u> ( <u>p<u>dn</u></u> = 0 to 7)
MTX    - <u>ALIAS</u> MTl <u>dn</u>	Any Magnetic Tape Unit ( <u>l<u>dn</u></u> = 0 to 7)

-WAIT is an optional parameter. If included, it holds the ASSIGN command if the device is assigned to another user. The assignment request remains held until the device becomes available or the user types the BREAK key (CONTROL-P) at the terminal; both occurrences return the user to PRIMOS. If the requested device is not available and the -WAIT parameter has not been included, the error message:

The device is in use. (ASSIGN)

will be printed at the terminal.

The use of each of these devices is discussed in the sections that follow.

After all I/O operations are completed, exclusive use is relinquished by the command:

UNASSIGN device

device is the same mnemonic used in the ASSIGN command except for magnetic tape. With magnetic tape, device is either "MTpdn" or "-ALIAS MTldn".

READING PUNCHED CARDS

Assign use of the parallel interface card reader by:

ASSIGN CR[n] -WAIT

n is the card reader number and may be either 0 or 1. If n is omitted, 0 is assumed.



To read cards from the card reader, load the card deck into the device and enter the command:

```
CRMPC deck-image [-PRINT] { -CR0 }
                        { -CR1 }
```

<u>deck-image</u>	The pathname of the file into which the card images are to be loaded.
-PRINT	Print card while reading.
-CR0	Use device CR0 (default).
-CR1	Use device CR1.

The CRMPC command translates the card images into an ASCII file. Cards are expected to be in 029 (EBCDIC) representation. Control cards may be inserted into the card deck to instruct the card reader, as follows:

<u>Columns 1 and 2 of deck control card</u>	<u>Instruction</u>
\$6	Placed before a deck of cards in 026 (BCD) format. Instructs the card reader to interpret 026 cards as if they were in 029 format.
\$9	Instructs the card reader to resume reading in 029 format.
\$E	Placed last in the deck and signals the end of the deck. Control returns to PRIMOS and the file is closed.

If the card deck is exhausted but contains no \$E card at the end (or if the reader is halted by the user), control returns to PRIMOS. However, the file is not closed. If more cards are to be read into the file, the reader should be reloaded; reading is resumed by the command START at the terminal.

Close the file with the command:

CLOSE -ALL

or

CLOSE deck-image



Example of card reading session:

OK, ASSIGN CR -WAIT  
 OK, CRMPC old-program-1  
 OK, UNASSIGN CR0  
 OK,

READING PUNCHED PAPER TAPE

First load the tape into the reader; then assign the tape reader. Source programs punched on paper tape in ASCII representation can be read into a disk file with the Editor.

OK, <u>ASSIGN PTR -WAIT</u>	Assign tape reader
OK, <u>ED</u>	Invoke Editor
<u>INPUT</u>	
(CR)	Switch to EDIT mode
<u>EDIT</u>	
<u>INPUT (PTR)</u>	Input from tape reader
<u>EDIT</u>	Tape is being read
<u>FILE filename</u>	File input under <u>filename</u>
OK, <u>UNASSIGN PTR</u>	Unassign tape reader

MAGNETIC TAPE OPERATIONS

The Prime magnetic tape utilities (MAGNET, MAGRST, and MAGSAV) allow the transfer of files from disk to tape and vice-versa, and the transfer and translation of tapes in selected non-Prime formats to and from PRIMOS disk files. All magnetic tape operations done with these utilities, as well as any user program written to use magnetic tape, require the assignment of at least one magnetic tape drive unit.

Assigning Tape Drives

Magnetic tape drive assignment can be set up at each installation by the System Administrator in one of three ways:

- Each user can assign a tape drive from any terminal; operator intervention is necessary only for processing special requests. This is the default mode.
- Each user must send all assignment requests through the operator, who controls all access to tape drives. The operator then sends messages to the user terminal indicating the status of the assignment request.



- Tape drive assignment from any user terminal is strictly forbidden. This feature is used to restrict access to tape drives in security-conscious environments, or when the operator is not available to process requests.

### The ASSIGN Command Format

Users may assign tape drives themselves, or they may request the operator to assign a drive for them.

User Assigned Drives: Users may assign tape drives directly in two ways:

- by physical device number (pdn):

ASSIGN MTpdn [options]

- by physical device number with a logical device number (ldn) as an alias:

ASSIGN MTpdn -ALIAS MTldn [options]

In the first of these ways, you assign the drive by physical device number (pdn), which requests that particular drive. A logical device number (ldn), which you will use in subsequent operations, is also automatically given to the drive. In this case, ldn equals pdn.

In the second of these ways, you request that the physical device number specified be assigned a given logical device number. The -ALIAS option supplies the ldn.

Use of certain options (such as -DENSITY) with either of the above formats may require operator intervention.

Operator Assigned Drives: Users may also request the system operator to assign a drive as follows:

- by using the MTX argument and a logical device number (ldn) as an alias:

ASSIGN MTX -ALIAS MTldn [options]

Using "MTX" always requires operator intervention and says, "Give me any tape drive, and call it number ldn". The -ALIAS option supplies the number. The options that you specify (if any) may instruct the operator to assign you a drive that can handle a particular type of tape (for example, a 9-track tape at 6250 bpi). Other options may make special requests to the operator, as for example, to remove the write-ring or to mount a particular tape.



Whether you or the operator assigns the drive, you will be told which physical device has been assigned to you. In your subsequent tape dialogs, you must refer to the device by ldn, except for UNASSIGN. (When you use UNASSIGN to release a drive, you may use either ldn or pdn.)

With any of the above three formats, -WAIT will hold the request if the appropriate drive is busy.

ASSIGN Options: The full command format for using ASSIGN with magnetic tape drives is:

```
ASSIGN { MTpdn [-ALIAS MTldn] } [options]
      { MTX   -ALIAS MTldn  }
```

The arguments and options are:

<u>Argument/Option</u>	<u>Description</u>
MTpdn	Magnetic tape (MT) unit number from 0 to 7, inclusive. <u>pdn</u> is the physical device number assigned to each drive at system startup. Numbers can be obtained from the system operator.
MTX	Tells the operator to assign "any available drive"; MTX <u>must</u> be accompanied by -ALIAS MTldn, which assigns a number (alias) to the drive for reference purposes. See below. The actual drive assigned depends on any other options which appear on the command line.
-ALIAS MTldn	The logical drive number, from 0 to 7, inclusive. <u>ldn</u> is a user-specified number assigned to a particular physical drive unit; used as an alias for the <u>pdn</u> in subsequent magnetic tape operations.
-WAIT	Indicates user is willing to wait until requested drive is available.
-TPID id	Requests the operator to mount a particular reel of tape, identified by a tape <u>id</u> ; requires operator intervention. <u>id</u> is a list of tape identifiers (arguments) describing a particular reel of tape, and/or type of tape drive (name, number, etc.). In general, identifiers should not begin with a hyphen (-), which is a reserved character indicating the next control argument on the ASSIGN statement line. (If an identifier is enclosed in single quotation marks, it may start with a hyphen or contain embedded spaces.)



{ -RINGON } { -RINGOFF }	Read and write both permitted. Read only; write protected in effect.
	Requires operator intervention for removal or replacement of write-ring.
-DENSITY { 800 } { 1600 } { 3200 } { 6250 }	Particular tape density settings are requested with this option. Requires operator intervention for drives that handle 800 and 1600 bpi (bits per inch) settings.
{ -7TRK } { -9TRK }	Indicates 7- or 9-track tape drive; default is 9-track. Requires operator intervention.
-MOUNT	Indicates that a new tape is to be placed on a tape drive already assigned. Requires operator intervention.

#### Using the -ALIAS Option

The -ALIAS option is useful in several general situations:

- When you request special features and do not know which available drive meets the stated requirements.
- When you are writing a command file to perform magnetic tape operations and have no way of knowing which tape drive is available at a given time.
- When you know the actual pdn of the drive being assigned but prefer to give it another number, for ease of reference, or to avoid confusion.

Once an alias has been assigned, the logical device number must be used to refer to the drive in question in subsequent magnetic tape drive operations like MAGSAV. The logical device number is "mapped into", or associated with the physical device number in an internal table.

With the MTX option, command files which perform magnetic tape operations can be executed independently of a particular drive's availability. The arbitrary number assigned the tape drive with MTX -ALIAS can be used in writing responses to the dialog of the utility invoked by the command file.



Note

MAGSAV and MAGRST ask the user for the device number of the drive on which a tape is mounted. Both dialogs assume the number given is a logical device number: consequently, the internal list of logical device numbers is searched first. If a match is found, MAGSAV/MAGRST will interact with the tape mounted on the corresponding physical drive. Suppose the user first assigns physical device MT0 as logical MT1, then assigns physical MT1 as logical MT0. If the user answers "1" to the "TAPE UNIT:" prompt of MAGSAV (or MAGRST), the utility assumes that "1" is a logical device number (ldn). Thus, it attempts to read from or write to, as the case may be, the tape mounted on physical device MT0, which the user previously assigned as logical MT1.

The STATUS DEVICE Command

The STATUS DEVICE command allows you to see which physical devices (tape drives) are currently in use. The output displays physical and logical device numbers for any assigned magnetic tape drives. Tape drives which are not assigned are not printed in the output. The format for the STATUS DEVICE command line is:

STATUS DEVICE

A typical display might be:

OK, STATUS DEVICE

Device	User name	Usrnum	Ldevice
MT0	PRANCER	13	MT0
MT1	DANCER	69	MT2

OK,

The display headings have the following meanings:

HeadingMeaning

Device	The physical device number.
User name	The login name of the user who has the device assigned.
Usrnum	The user number of that user.
Ldevice	The logical device number being used for the drive.



## USING ASSIGN

The following examples illustrate some uses of ASSIGN. In all cases, the distinction between what the user can do without operator intervention and what must be done with operator assistance is indicated.

### Default Assignment

The standard form of assignment does not require operator intervention on systems with the default configuration (user-privileges allowed). For example:

```
OK, ASSIGN MT1
Device MT1 Assigned.
```

Magnetic tape drive MT1 is assigned. (1 is the physical device number, and also, in this case, the logical device number.) If the device is currently assigned to another user or process, this message appears:

```
The device is in use. MT1 (asrmt$)
ER!
```

On systems where all magnetic tape requests are monitored, the request above would be acknowledged with the same message, but a slight delay would be observed. The operator has to approve each request, which results in a delayed response at the user terminal.

### Logical Aliases

Logical device numbers can be assigned by the user without operator assistance on default privilege systems, providing that no other special requests are made on the same ASSIGN command line:

```
OK, ASSIGN MT1 -ALIAS MT0
Device MT1 assigned.
```

Note that the physical, not the logical, device number is returned.

Physical device MT1 can now be referred to as logical device MT0. If no ldn alias is requested, the default logical device number is the same as the physical device number of the drive. The STATUS DEVICE command lists the physical-to-logical number correspondence:

```
OK, STATUS DEVICE
```

Device	User name	Usrnum	Ldevice
MT1	JOHN	69	MT0

```
OK,
```



If no logical alias had been requested, the Ldevice entry would be identical to the Device entry; in this case, MT1.

### Aliases in Operator Mode

Similarly, logical aliases can be requested on operator-controlled systems. Again, the pdn of the assigned device will be displayed at the user's terminal with a message of this general form:

Device MTpdn assigned.  
pdn varies with the actual physical device chosen by the operator.

### Special Requests

If control arguments for special requests appear on the ASSIGN command line, then the operator must intervene, even on systems with default user privileges. For example, all ASSIGN commands with the MTX option must be handled by the operator:

ASSIGN MTX -ALIAS MT4

The operator is requested to assign any available tape drive as logical device 4. A message is displayed at the user's terminal, indicating which physical drive has been assigned.

The operator must also intervene if a user wants a tape mounted, or if a particular density setting is required, or if a particular drive is needed (for instance, to read a tape recorded at 6250 bpi). For example:

ASSIGN MTX -ALIAS MT3 -TPID POWER -9TRK -RINGOFF -DENSITY 6250

The operator is requested to mount the "POWER" tape on a 9-track drive that can handle 6250 bpi. In this case, "POWER" is the name written on the tape reel to identify the tape and is not necessarily the recorded label. In addition, the user wants write-protection and is assigning an alias of MT3 (ldn) to whatever device the operator chooses. This request, if processed, might be acknowledged with this display:

Device MT0 assigned.



### Operator Not Available

If the operator is not available to handle requests, any attempt by a user to assign a mag tape drive will result in this message:

```
OK, ASSIGN MT1
No Magtape assignment permitted. (asrmt$)
ER!
```

### Operator Can't Handle

If any request cannot be handled by the operator for any reason, the following message appears at the terminal:

```
OK, ASSIGN MTX -ALIAS MT0 -DENSITY 3200
Magtape assignment request aborted (asrmt$)
ER!
```

### Summary of ASSIGN Command Messages

Messages you get from using the ASSIGN command are either informational or error-related. These messages are listed and described here alphabetically.

- Bad parameter. (asrmt\$)

This message indicates an ASSIGN syntax error; your input is invalid.

- Device MTpdn assigned.

This is an informational message letting you know that your physical device has been assigned.

- Device not assigned. MTn (asrmt\$)

This error message indicates that the device you specified has not yet been assigned.

- Improper command usage or arguments. (ASSIGN)  
ER!

This error message indicates an ASSIGN syntax error; your input is invalid.



- Magtape assignment request aborted (asrmt\$)  
ER!

This is an error message indicating that the operator cannot handle your assignment request.

- No magtape assignments permitted. (asrmt\$)  
ER!

This error message indicates that the operator is not permitting any magnetic tape assignments at this time.

- The device is in use. MTn (asrmt\$)  
ER!

This error message indicates that another user has the specified device assigned.

#### RELEASING A TAPE DRIVE (UNASSIGN)

When a user completes a magnetic tape operation, the magnetic tape drive should be released for general use. Simply issue the UNASSIGN command with one of the indicated arguments:

```
UNASSIGN { MTpdn } [-UNLOAD]
          { -ALIAS MTldn }
```

The -ALIAS option can be used to unassign a drive whether or not the user assigned an alias to the drive. The ldn argument value can be either the user-chosen logical device number, if one was assigned, or the default ldn (which is identical to the pdn), when -ALIAS was not used in assigning the drive.

The -UNLOAD option normally rewinds the tape and places it off-line. However, on some drives or controllers -UNLOAD only rewinds the tape. Check with your System Administrator for the action provided by -UNLOAD at your installation.

#### Who Can Unassign a Drive

A tape drive can be unassigned only by:

- The user who assigned it (on default-privileged systems)
- The system operator



The system operator can unassign any drive using the pdn argument; the "-ALIAS ldn" option can be used only if the drive is owned by (that is, was previously assigned by) the user.

If an operator unassigns your tape drive, no message will appear at your terminal. Should you subsequently attempt to unassign the same device, an error message will be displayed.

For example, suppose that you assign the following tape drives:

```
OK, ASSIGN MT1 -ALIAS MT2
Device MT1 assigned.
OK, ASSIGN MTX -ALIAS MT0
Device MT0 assigned.
OK,
```

You can unassign physical drive MT1 with either of the following:

```
UNASSIGN MT1
UNASSIGN -ALIAS MT2
```

You can unassign physical drive MT0 with either of the following:

```
UNASSIGN -ALIAS MT0
UNASSIGN MT0
```

The operator can unassign these drives only with the following commands:

```
UNASSIGN MT1
UNASSIGN MT0
```

#### UNASSIGN Command Messages

Messages you get from using the UNASSIGN command are either informational or error-related. All UNASSIGN command messages are listed and described here alphabetically.

- Bad parameter. (usrmt\$)

This message indicates an UNASSIGN syntax error; your input is invalid.

- The device not assigned. MTn (usrmt\$)

This error message indicates that the device you are attempting to unassign has not yet been assigned.



- Device released.

This is solely an informational message telling you that the device you unassigned is released.

### MAGNETIC TAPE UTILITIES

Three of Prime's magnetic tape utilities include:

MAGNET	Allows you to transfer both Prime-format and non-Prime-format files and tapes from tape to disk, disk to tape, and tape to tape.
MAGRST	Restores Prime-format files, directory-trees, or disk volumes saved by MAGSAV from tape.
MAGSAV	Archives Prime-format files, directory-trees, or disk volumes to tape.

These utilities are described briefly below. Complete information is contained in the Magnetic Tape User's Guide.

### THE MAGNET UTILITY

To transfer data by magnetic tape from a non-Prime operating system to PRIMOS and vice versa, you must write the magnetic tape in an interchange format that both operating systems can understand. MAGNET is the PRIMOS subsystem that allows you to read and write magnetic tapes in these various interchange formats. MAGNET is intended primarily for users who have had prior experience with magnetic tapes.

Some of the facilities that the MAGNET subsystem provides are:

- Declaration and modification of I/O objects
- Linkage of I/O objects with PRIMOS global variables
- Translation to and from various character sets
- User-definable character sets
- Seven-track binary packing and unpacking
- Tape positioning
- Physical tape copying



- Logical data transfer between devices with support of multiple destinations
- Support of unlabelled tapes and of ANSI and IBM standard labelled tapes
- Support of fixed-length records and of IBM, ANSI, and Prime variable-length records
- Support of old (pre-rev 18.4) MAGNET features for the READ, WRITE, COPY, and POSITION subcommands
- Support for tape operations within Batch mode (operator intervention)
- Tape-to-spooler and disk-to-spooler support
- Break key handling

Instructions on how to use MAGNET are contained in the Magnetic Tape User's Guide.

#### SAVING DISK FILES ON TAPE (MAGSAV)

The Magnetic Tape Save Utility (MAGSAV) copies PRIMOS file system objects from disk to seven-track or nine-track magnetic tape. Before invoking MAGSAV, you must first use ASSIGN to assign your magnetic drive. You invoke MAGSAV in the following format:

MAGSAV [options]

The following options may be specified on the MAGSAV command line:

- |                        |  |
|------------------------|--|
| -7TRK                  | Uses seven-track magtape format. The default is nine-track.  |
| -INC                   | Indicates incremental dump. Only file system objects with DUMPED switch set to 0 will be saved. If you do not specify -INC, all objects are saved. |
| { -SAVE_UFD<br>-SUFD } | Tells MAGSAV to save all directories whether or not they have been modified. (-SAVE_UFD is used only with the -INC option.)                        |
| -LONG                  | Sets record size to 2048 bytes. The default record size is 4096-byte variable length records.  |
| -P300                  | Specifies 1024-byte records and suppresses ACLs.   |
| -UPDT                  | Indicates update. The DUMPED switch is set for files and directories saved from disk to tape.  |



-TTY MAGSAV takes tape unit number from terminal and all other information from current input stream.

{ -NO\_ACL } Specifies that MAGSAV is not to save any ACLS or ACL  
 { -NOA } references. Tapes saved with the -NO\_ACL option can be restored by Rev. 18 MAGRST onto a Rev. 18 system. If -NO\_ACL is not specified, ACL information for objects protected by specific ACLs or access categories (but not by default protection) is saved to tape. ACLs are explained in Chapter 16.

### MAGSAV Dialog Summary

The MAGSAV dialog is summarized below. Suggested user responses are indicated.

<u>Prompt</u>	<u>Response</u>
Tape unit (9 Trk):	Enter logical tape drive number, from 0-7. If the -7Trk option was not specified, (9 Trk) is displayed.
Enter logical tape number:	Enter number, from 1 to n, of desired logical tape (see Note at the end of this list). The tape is then rewound and positioned. Specify 0 if tape is already positioned as desired. "0" will position the tape after the last recorded object.
Tape name:	Specify a name or identifier for this tape; maximum of 6 characters.
Date (MM DD YY):	Specify date in format: mm dd yy. Default (CR) is system-supplied date.
Rev no:	Enter arbitrary number, or (CR).
Name or Command:	Possible responses include: <div style="margin-left: 40px;"> <u>filename</u> Saves file or directory named.  <u>dirname</u>  <div style="margin-left: 40px;">MFD Saves entire disk volume. (You must be attached to an MFD to use this.)</div> <div style="margin-left: 40px;">* Saves current directory.</div> </div>



`$A ufd [password] [ldisk] [key]:` Changes home UFD to ufd. If ldisk number is not specified, all disks are searched for ufd (default). Supply a numerical value for key to attach to a sub-UFD. A value of 2, for example, will attach you downward one level. If you wish to attach to a sub-UFD in the current UFD, then set ldisk equal to 0 and key to 2. `$A` does not accept pathnames.

`$I [filename] n:` Prints at your terminal an index of files and directories saved from disk to tape. The index is written to a file if a filename is provided. n indicates number of levels in tree structure hierarchy to be included in the index. (The default is two levels.) `$I` does not accept pathnames.

`$Q` Terminates logical tape and returns to PRIMOS.

`$R` Terminates logical tape, rewinds tape and returns to PRIMOS.

`$INC`

ON	Turns incremental save option on or off; same as -INC command line option, above.
OFF	

#### Note

A "logical tape" results from single invocation of MAGSAV. It is a unique entity, with its own header, and so on. It may be a portion of a physical tape, or a complete physical tape; or it may span one or more physical tapes. A single physical tape may contain several logical tapes, each of which is identified by number.

When an unrecoverable tape error is encountered with MAGSAV, you will get an appropriate error message and will be requested to specify a new logical tape number. If recoverable errors occur, MAGSAV will tell you at the end of your dialog how many errors it has recovered.



Sample MAGSAV Session

Below is an example taken from a terminal session during which a disk file (TAPE.EX) was saved on tape. If a carriage return (CR) is given in response to the "Date" and "Rev no" prompts, as shown below, the system will supply the current date and software revision number. The logical device number (ldn) is supplied as the response to the "Tape unit" prompt. If you did not use the -ALIAS option when you assigned the tape drive, the logical device number (ldn) is the same as the physical device number (pdn).

OK, ASSIGN MT1 -ALIAS MT7

Device MT1 assigned.

OK, STATUS DEVICE

Device	User name	Usrnum	Ldevice
MT1	ANITA	67	MT7

OK, MAGSAV

[MAGSAV Rev. 19.0]

Tape unit (9 Trk): 7

Enter logical tape number: 0

Tape name: MAGTAP

Date (MM DD YY):(CR)

Rev no:(CR)

Name or Command: TAPE.EX

Name or Command: \$Q

OK,

How MAGSAV Handles ACLs

A file or directory using ACL protection can be protected in one of three ways: it can be protected by a specific ACL; it can be protected by an access category; or it can use the default protection of the directory in which it resides. (ACLs are explained in Chapter 16.) MAGSAV handles ACLs in the following manner:

- Specific ACLs are always saved (unless the user gives the -NO\_ACL option).
- If a UFD is saved, all access categories within the UFD are saved.
- If individual files are being saved by name, then any desired access categories must also be saved by name. They are not saved automatically, as specific ACLs are.
- No ACL information is saved for file system objects that use default protection.



### MAGSAV and Quotas

When MAGSAV saves a UFD, it also saves any quotas set on directories. (Quotas are explained in Chapter 17.)

### MAGSAV Messages

The MAGSAV utility issues informational and error-related messages to the user. These messages are listed and explained in Chapter 8 of the Magnetic Tape User's Guide.

### RESTORING FILES FROM TAPE TO DISK (MAGRST)

The Magnetic Tape Restore Utility (MAGRST) restores Prime format files, directories, trees, and partitions from a magnetic tape (seven-track or nine-track) to a disk. All information is restored to the directory to which the user is currently attached. MAGRST can read tapes of any record size, with fixed-length or variable-length records (up to 4096 bytes), making it compatible with MAGSAV.

The command format is:

MAGRST [-7TRK] [-TTY]

<u>Option</u>	<u>Function</u>
-7TRK	Specifies seven-track format. The default is nine-track.
-TTY	Takes the tape unit number from your terminal, even if all other information is taken from the current input stream. You use this option with CPL files and command input files, but <u>not</u> with files meant to run as phantoms or Batch jobs.

### MAGRST Dialog Summary

The MAGRST utility displays a series of questions and messages which are summarized below, along with appropriate responses and descriptions.



<u>Prompt/Message</u>	<u>Response/Description</u>
You are not attached to an MFD	This message is mainly a warning for the operator doing system backups. In this case, the operator <u>does</u> want to be attached to the MFD. Users will <u>not</u> normally be attached to the MFD; they will be attached to the directory to which they want objects restored. Therefore, users will normally get this message and can ignore it.
Tape unit (9 Trk):	Enter logical device number; from 0-7. The (9 Trk) message is displayed if the -7Trk option was not specified on the MAGRST command line.
(Tape not at load point)	This message appears if the tape is not positioned to the beginning of the tape.
Enter logical tape number:	If tape is divided into several logical units, enter logical tape number from 1 to n. Tape is positioned to specified logical tape. Enter 0 if tape is already positioned as desired. (No action is taken in this case.) See the <u>Note</u> at the end of this list.
Name: tape-name	MAGRST displays the name of the logical tape currently positioned to; names are provided during MAGSAV dialog.
Date (MM DD YY): tape-date	MAGRST displays date on tape was recorded. Supplied during MAGSAV.
Rev no: number	MAGRST displays arbitrary <u>number</u> specified during MAGSAV.
Reel no: reel-number	MAGRST displays appropriate <u>reel-number</u> of tape.
Ready to Restore:	Enter one of the following options:  <u>YES</u> : Restores entire logical tape and returns to PRIMOS.  <u>NO</u> : Causes first prompt ("Tape unit") to be reissued.



\$I [filename] n: Prints tape index to n levels at terminal during restore. Index can be optionally saved to indicated filename. \$I does not accept pathnames.

NW [filename][n]: Prints n level index at terminal but DOES NOT UPDATE disk because no files are restored. Optionally stores index in filename. NW does not accept pathnames.

PARTIAL: Restores only certain files and directories. Pathnames are entered in response to "TREE NAME:" prompt.

\$A ufd [password] [ldisk] [key]: Changes home UFD to ufd. If ldisk number is not specified, all disks are searched for ufd. Supply a numerical value for key to attach to a sub-UFD. A value of 2, for example, will attach you downward one level. If you wish to attach to a sub-UFD in the current UFD, then set ldisk equal to 0 and key to 2. \$A does not accept pathnames.

Tree name:

This prompt is returned when PARTIAL option is specified. Respond with one of the following:

pathname: Names file or directory to be restored. pathname should not include name of directory to which user was attached when saving file or directory, except when attached to an MFD. For example, if a file, file2, was saved from the UFD TOP, and its pathname is: TOP>MID>file2, it can be restored with the pathname: "MID>file2", but NOT with the pathname: "TOP>MID>file2".

(CR): Terminates MAGRST dialog by indicating end of treename list; tape is read, and control returns to PRIMOS.



Note

A "runaway" tape condition can occur if there is only one logical tape on the currently mounted reel of tape and the user specifies a number greater than 1 in response to the LOGICAL TAPE NUMBER prompt. If this happens, MAGRST will search endlessly for the non-existent logical tape(s) and will consequently be unable to read the end-of-tape marker. To abort the unsuccessful search, either the drive must be unassigned, or the off-line button on the drive must be pressed.

When an unrecoverable error is encountered during an attempted MAGRST operation, an error message is displayed. Recoverable errors are logged and a total is displayed when the end of the logical or physical tape is reached.

Sample MAGRST Session

The following example represents the dialog necessary to restore a file from tape to disk. The file saved in the previous MAGSAV sample session (TAPE.EX) is used in this example also.

```
OK, MAGRST
[MAGRST Rev. 19.0]
You are not attached to an MFD
Tape unit (9 Trk):0
Enter logical tape number: 1
Name: MAGTAP
Date(MM DD YY): 04-30-82
Rev no:      0
Reel no:     1
Ready to Restore: PARTIAL
Tree name: TAPE.EX
Tree name: (CR)
*** Starting restore ***
*** End logical tape ***
*** Restore complete ***
OK,
```



How MAGRST Handles Protection

At Rev. 19, users can choose between ACL and password protection for their directories. MAGRST can handle both types of protection.

If there is a conflict in a matter of protection between an object being restored from tape and an object of the same name resident on the disk, the protection on the disk is retained. The version on the disk is assumed to be the more recent one, and thus is more likely to reflect the owner's current wishes. This general rule is expanded below.

ACL vs. Password Directories: Whenever possible, MAGRST restores ACL directories as ACL directories and password directories as password directories. However, if a directory currently on the disk has the same name, but a different type, from a directory being restored from tape, the type of the directory on the disk will be maintained.

How MAGRST Handles ACLs: MAGRST always tries to restore the ACL protection saved by MAGSAV. The following exceptions exist:

- If an object being restored already exists on the disk, the protection on the disk is retained, and the protection on the tape is ignored.
- An access category is restored only if no access category of that name exists on the disk. (If an access category of the same name does exist, the user will be warned of that fact.)
- If a file protected by an access category has been saved by name or is restored by name, the access category is not restored. (In the former case, the access category may not even be on the tape.) If the access category already exists on the disk, then the object will be protected by that category. Otherwise, the object will receive default protection, and an error message will be printed.

Therefore, if you wish to restore specific files, together with their access categories, you should first restore the access categories, and then restore the protected files.

- If objects have been saved with the `-NO_ACL` option, MAGRST restores them with the same protection as the parent directory.

How MAGRST Handles Quotas

Quotas will be restored if the corresponding UFD does not already exist on the disk. If the UFD does exist, its existing quota will remain in effect.



### MAGRST Messages

The MAGRST utility issues informational and error-related messages to the user. These messages are listed and explained in Chapter 8 of the Magnetic Tape User's Guide.



# 14

## Using PRIMENET

### INTRODUCTION

Many Prime installations contain two or more systems connected in a network—a combination of communications hardware and PRIMOS software called PRIMENET. In a network, the system to which the user terminal is connected is the local system, while all other systems are considered remote.

On a system using PRIMENET, you can:

- Log in on a remote system and use that CPU for processing. (Only terminal I/O is sent across the network.)
- Log in to your local system, then ATTACH to directories on disk volumes connected to any other processor in the network, and access files in such directories. (File data is transmitted across the network; the local CPU does the processing.)
- Use a pathname with a command (such as COPY) or with a subsystem (such as the EDITOR) to access a file on a remote disk. For example:

```
COPY <FOREST>PINE>TWIG  
ED <FOREST>OAK>BRANCH5>ACORNLIST
```

- Use FTS (File Transfer Service) to transfer files between a local and remote site. (FTS is a separately priced product.)



- Establish a remote-id, necessary for security reasons on certain remote systems, that will enable you to access these systems.

### REMOTE LOGIN

Each processor in the network is assigned a systemname (also called a nodename) during system configuration. The systemname identifies the processor for remote logins. (Users can determine the systemnames of remote processors by using the STATUS NETWORK command, explained below.) The format for remote logins is:

LOGIN user-id [login-password] -ON systemname [-PROJECT project-id]

If -ON systemname is omitted, an attempt is made to log in as user-id on the local system only. If systemname is the name of the local system, the login attempt is done locally without the use of PRIMENET. If any of the systems on which you work uses specific project names, you must supply the appropriate project-id when you log in. Your administrator should give you any required project-ids when you receive your user id.

If the LOGIN command fails because the user id or password is invalid, the user's PRIMENET connection is broken. Input from the user's terminal is again processed by the local processor, but the user is not logged in. If a syntax error occurs (such as giving 2JOE as a user id), the PRIMENET connection is not broken, but the user is still not logged in.

On a terminal logged in to a remote processor, the command LOGOUT logs out the process, breaks the remote connection over PRIMENET, and reconnects the terminal to its local system (not logged in). The message:

Wait...

Disconnected from systemname  
OK,

is displayed. All input characters typed between the LOGOUT command and the response "OK," are discarded.

### Network Status

You may learn the names and states of all systems (or nodes) in the network by giving the command:

STATUS NETWORK



For example:

OK, STATUS NETWORK

Ring Network

Node	State
SYSE	****
SYSA	Up
SYSB	Up
SYSC	Up
SYSD	Up
SYSF	Down
SYSG	Up
SYSH	Down

OK,

This shows the state of a eight-system network as it would be printed for a local user on the SYSE system. The UP state means that the system is configured, connected, and functioning. The system to which you have logged in is given first and shown by asterisks (\*\*\*\*).

#### ATTACHING TO REMOTE DIRECTORIES

Attaching to a remote directory is the same as attaching to a local directory. You can give the name of the disk partition or logical disk number (determined from a STATUS DISKS display, illustrated below) within the ATTACH command, as in:

ATTACH <SHARK>JAWS

Alternately, you may omit the diskname as in:

ATTACH JAWS

When you omit the diskname, PRIMOS searches each logical disk beginning with logical disk number 0, and attaches you to the first UFD of that name it finds. PRIMOS searches all local disks first, then remote disks in the order in which they have been configured for the system on which you are working. You can discover this order with the STATUS DISKS command explained below.



The STATUS DISKS Command

You can discover the names and numbers of logical disks on both local and remote systems using the STATUS DISKS command. For example:

OK, STATUS DISKS

Disk	Ldev	Pdev	System
STATS	0	3462	
FIELDS	1	460	
MISCEL	2	71063	
FOREST	3	71061	
REEFS	4		SYSC
LAGOON	5		SYSD
SHARK	6		SYSD
SHARK2	7		SYSD
CLOUDS	12		SYSE
CLIFF1	13		SYSE
CLIFF2	14		SYSE
AERIE	15		SYSE
ROCK	23		SYSA
FALCON	24		SYSA
NEST1	25		SYSA
NEST2	26		SYSA

In the STATUS DISKS printout, "Disk" is the name of the logical disk. "Ldev" is the logical disk number. "Pdev" is the physical disk identifier, which appears only for partitions on the local system. "System" is the systemname. The disks that have no systemname listed are on the local system. Remember that the disk list gives you the order in which disks are searched for UFDs. This order can be important in a system or network where two UFDs have the same name. (See "UFDs with the Same Name", below.)

UFDs with the Same Name

On any given disk, UFD names must be unique. However, the same UFD name may appear on different disks in the same network. If the UFD you wish to access is the first one that PRIMOS will reach in its access search, you need not give the diskname in the pathname. If, however, the UFD you wish to access is not the first one with that name that PRIMOS will reach, you must give the diskname in the pathname, or PRIMOS will not attach you to the UFD you want.



Example: Assume you are logged in to the local system in the example given above for the STATUS DEVICE command. Two UFDs named QUARTZ exist in this network, one in the disk partition CLIFF2 and one in the disk partition ROCK.

To attach to QUARTZ in CLIFF2, you may give either of the following commands:

```
ATTACH QUARTZ
ATTACH <CLIFF2>QUARTZ
```

To attach to QUARTZ in ROCK, you must specify the diskname in the pathname as follows:

```
ATTACH <ROCK>QUARTZ
```

Otherwise, PRIMOS would attach you to QUARTZ in CLIFF2, since this is the first UFD of the name that it encounters as it searches all the disks in logical-disk-number order.

You may be attached anywhere in the network, and the search order will still be the same (from STATS to NEST2).

#### ACCESSING REMOTE SYSTEMS AND NETWORKS WITH NETLINK

You may connect to any system on your network using the NETLINK command. This means that if your system is part of a Public Data Network (such as TELENET®), non-Prime systems and non-PRIMENET software may be accessed. Other sites or other networks as well as jobs within these other sites and networks may be accessed.

Several NETLINK commands let you use these other systems and networks. There are basic commands and advanced commands. Basic commands allow you to enter and exit the remote systems. Advanced commands allow you to:

- Transfer files across networks
- Set data transmission characteristics
- Print the status of your connection
- Connect to and use up to four different remote systems at the same time
- Specify the various fields of the connect packet when data transmission characteristics of a foreign system differ from those of Prime's.

Only NETLINK's basic usage will be presented here. For a list of all NETLINK commands and error messages see the PRIMENET Guide.



### Basic NETLINK Usage

The basic steps to using NETLINK are as follows:

1. Enter NETLINK Command Mode by issuing the command:

NETLINK

When Command Mode is entered, the @ prompt appears.

2. Connect to the remote system by issuing the subcommand:

C address

address is either the host address assigned by the Public Data Network or a PRIMENET system name. For example, "906 23" and "SYS2" are both valid addresses.

When a connection has been established, the following message appears:

address Connected

3. Login to the system as you would normally, entering any validation codes or passwords as required.
4. Once you finish a terminal session, logout as you would normally. The following message appears:

address Disconnected

When a connection to a remote host has been terminated by logging out, Command Mode is re-entered and the @ appears. You may now connect to another site or return to PRIMOS.

5. To return to PRIMOS enter the command:

QUIT



# NETLINK Example

Below is an example of a basic terminal session. User responses are underlined.

OK, NETLINK  
[NETLINK Rev. 19.0]

@ C SYS2  
SYS2 Connected

PRIMENET 19.0 SYS2  
LOGIN HOBBIT

HOBBIT (user 49) logged in Wednesday, 31 Mar 82 11:21:33.  
Welcome to PRIMOS version 19.0.  
Last login Wednesday. 31 Mar 82 9:46:24.

Enter validation code: SHIRE [Validation is installation  
OK, dependent; code, if required, does  
not print on user's terminal.]  
.  
.  
.  
[Continue with normal terminal  
session.]

OK, LOGOUT

HOBBIT (user 49) logged out Wednesday, 31 Mar 82 13:07:40.  
Time Used: 01h 46m connect, 03m 15s CPU, 04m 12s I/O.

Wait...

SYS2 Disconnected

@ QUIT

OK,

## Alternate NETLINK Usage and Example (The -TO Option)

You may use the option:

-TO address

on the NETLINK command line, which will bypass NETLINK's @ prompt asking for the "C address" response (discussed above under Basic NETLINK Usage). NETLINK will connect you directly to the address. You can then log in, do your work, and log out. After logout you will return directly to your original system, without getting the @ prompt or needing the "QUIT" reply.



For example:

```
OK, NETLINK -TO SYSX
[NETLINK Rev. 19.0]
```

```
SYSX Connected
PRIMENET 19.0 SYSX
LOGIN GRIFFIN
```

```
GRIFFIN (user 26) logged in Tuesday, 20 Apr 82 17:55:40.
Welcome to PRIMOS version 19.0.
Last login Tuesday, 20 Apr 82 17:53:20.
```

```
Enter validation code: TOAD [Again, code does not print.]
```

```
OK,
```

```
      .
      . [Continue with normal terminal
      . session.]
```

```
LOGOUT
```

```
GRIFFIN (user 26) logged out Tuesday, 20 Apr 82 17:56:12.
Time used: 00h 42m connect, 01m 06s CPU, 02m 17s I/O.
```

```
Wait...
```

```
SYSX Disconnected
```

```
OK,
```

### The -MODE REMOTE\_ECHO Option

If you use Prime's Office Automation System (OAS) or the EMACS screen editor remotely (both separately priced products), you may also wish to use the -MODE REMOTE\_ECHO option to the NETLINK command. This option enables you to receive screen echoes from OAS and EMACS faster than would otherwise be the case.

You enter the option in one of two places:

- on the NETLINK command line, as in:

```
OK, NETLINK -TO SYSZ -MODE REMOTE_ECHO
```

- following the @ prompt, as in:

```
@ C SYSZ -MODE REMOTE_ECHO
```

This option is explained more fully in the PRIMENET Guide.



### NETLINK's HELP Facility

You can get a brief summary of NETLINK subcommands and options by specifying HELP in response to the @ prompt. For example:

```
OK, NETLINK
[NETLINK Rev. 19.0]
```

```
@ HELP
```

Following the HELP display, you will receive another @ prompt and can continue with your NETLINK session. More information on NETLINK's HELP facility appears in the PRIMENET Guide.

### TRANSFERRING FILES BETWEEN SYSTEMS (FTR)

#### Introduction

The FTR (File Transfer Request) command provides a method of transferring files between Prime computers that are connected via PRIMENET links. The FTR command is part of the File Transfer Service (FTS), which is a separately priced product.

To transfer a file, you submit a request that details all the necessary information for the transfer to occur. You can make a request even when the communications link between the two relevant computers is not operational or the remote computer is down, because requests are queued on the local (requesting) computer.

Once you have submitted a request for a file transfer, you may display or cancel the request. The following sections explain these operations briefly. Further information is available in the PRIMOS Commands Reference Guide and in the PRIMENET Guide.

#### Note

At present only SAM and DAM type files can be transferred. Specifying any other type of file system object (such as a directory, segment directory, or access category) will cause the transfer to fail.



Access: In order for full FTS to work correctly (including options discussed in the PRIMENET Guide), you, as a user, need the following rights:

Access Needed by Your User Id

For Sending Files

LURW access to  
source directory

For Fetching Files

ALURW access to  
destination directory

In addition, the "File Transfer Service (FTS) server" needs access rights. The FTS server is a special phantom that performs your FTS work for you. The server needs the following rights:

Access Needed by FTS Server

<u>Directory</u>	<u>Access</u>
Directory containing source file	DALURW
Directory to receive file	DALURW
All higher directories in source and destination pathnames	U

The letters in the charts above refer to the access codes used by access control lists, which are explained in Chapter 16. The letters have the following meanings: delete (D), add (A), list (L), use (U), read (R), and write (W).

The user id of the FTS server is set by your System Administrator. Your Administrator can tell you what it is so that you can grant it the appropriate access rights.

If you wish to keep your main directory private, you may wish to create a special sub-directory for your transfers and give the FTS server the necessary rights to this directory only.



Source and Destination Sites: File transfers take place between sites. A site (also called a system or node) is a single computer, identified by a unique site name; Prime sites normally use their PRIMENET system names as site names. Files are transferred from a source site to a destination site. One of these must be your local site; the other is usually a remote site. (FTR cannot be used to transfer files between two remote sites.)

The following discussion assumes that you are using FTR between Prime machines that have been configured using the FTGEN command, explained in the System Administrator's Guide. To transfer files to or from sites which are not configured, see the PRIMENET Guide.

Request Names and Request Numbers: Each request has two means of identification associated with it; a request name and a unique request number. FTR assigns the number when you submit the request. You can use either the name or the number to identify the request. The name is either the name of the file to be transferred, or a specific name that you assign to the request when you submit it. You usually refer to the request by its name. You may use the unique request number to distinguish between two requests that have the same name.

#### Submitting a File Transfer Request

Sending a File: To send a file to another Prime computer, use the FTR command with the following format:

```
FTR sourcefile destinationfile -DSTN_SITE sitename
```

Abbreviation for -DSTN\_SITE: -DS

sourcefile is the pathname of the file to be sent. You may give a filename if the file is in your current directory.

destinationfile, also a pathname, specifies the name the file will be given at the destination site after it has been transferred. Directories specified in the destination pathname must already exist for the transfer to work. (They will not be created by FTS.)

#### Note

If the pathname for a source file or destination file contains directory passwords, the passwords must be included in the pathname. The whole pathname must be included in quotation marks. For example:

```
'MARPLE CLUE>EVIDENCE'
```

-DSTN\_SITE sitename specifies the name of the destination site.



FTR will give the following response to your request:

Request request-name (request-number) submitted.

request-number is the unique identification number assigned by FTR.

For example, assume you are on SYS2. The following dialogue illustrates how to submit a request:

```
OK, FTR 'CENTER KEY>REPORT' EXPOST>GROUP2>TEXT -DS SYS4
[FTR rev 1.0]
Request REPORT (65211149) submitted.
OK,
```

In this example, FTR queues a copy of the file REPORT in the UFD CENTER (with the password KEY) to send to system SYS4, for deposit in the directory EXPOST>GROUP2 under the name TEXT. The request name is the source filename, REPORT. The unique request number is 65211149.

Fetching a File: To fetch a file from another Prime computer, give the command:

```
FTR sourcefile destinationfile -SRC_SITE sitename
```

Abbreviation for -SRC\_SITE: -SS

sourcefile and destinationfile are used as defined above for sending a file.

-SRC\_SITE sitename specifies the name of the site where the file to be fetched is stored.

For example, assume you are on SYS2. The command:

```
FTR PEOPLE>LIST MYUFD>MYLIST -SRC_SITE SYS6
```

will copy the file LIST in the UFD PEOPLE on SYS6 into the UFD MYUFD on SYS2 under the name MYLIST. The request name is the source filename, LIST.

Printing a File at a Remote Site: To print a hard copy of a file on a printer at another site, use the following format:

```
FTR sourcefile -DSTN_SITE sitename -DEVICE LP -DSTN_USER name
```

Abbreviation for -DSTN\_USER: -DU

-DEVICE LP is the option that instructs FTR to print the file on a line printer at the remote site specified by -DSTN\_SITE sitename.



`-DSTN_USER` name specifies the name of the person who should receive the printout at the remote site. The file will be printed with the name of the FTS server (set by your System Administrator) on the first line of the banner, where your user-id normally appears. The name specified after `-DSTN_USER` will appear on the second line of the banner of the printed file. name need not be a user-id.

For example, assume you are on SYSA. The command:

```
FTR STUART>LETTER -DSTN_SITE SYSF -DEVICE LP -DSTN_USER JUDY_JONES
```

will cause the file LETTER to be printed (at a line printer) on SYSF. The first line of the banner will be the name of the FTS server; the second line will be JUDY\_JONES. The request name is LETTER.

### Checking the Status of Requests

Brief Reports: Once you have submitted a request you can check its status with the command:

```
FTR -STATUS [request-name  
            [request-number]
```

The `-STATUS` option returns a brief, one-line report for each of your file transfer requests identified by request-name or request-number. If you omit the request name or number, you will receive a report on all of your current requests. The report has the following form:

```
date.time user-id request-name (request-number) Status -status.category
```

date.time is in the form YY-MM-DD.HH:MI:SS. status.category will be one of the following:

```
waiting
transferring
put on hold by user
put on hold by operator
```

"waiting" indicates that the request is in the transfer queue, but has not yet been transferred. "transferring" indicates that the transfer is in progress. The "hold" statuses indicate that a request is being indefinitely held in the transfer queue by command of either the user or operator. (How to hold files is explained in the PRIMENET Guide.)

For example,

```
OK, FTR -STATUS
[FTR rev 1.0]
82-03-30.10:52:12 ELLEN CHAPTER (36949288) Status -waiting
OK,
```



In this example, ELLEN has one request, named CHAPTER, waiting to be transferred.

Full Reports: You can ask for a full report on the status of your requests, with the command:

```
FTR -DISPLAY [request-name
              request-number]
```

Specifying request-name will print full information on all current requests with this name. Specifying request-number will print full information on the request with this number. If you omit request-name and request-number, FTR will print detailed information on all of your current requests.

The display takes the following form:

<u>Category</u>	<u>Information on the Request</u>
Request	Request-name (request-number)
User	User-id of submitter
Queue	Queuenam where the request is queued
Queued	Date.time queued; Status - waiting, transferring, put on hold by user, or put on hold by operator
Last attempt	Date.time of transfer attempt; Attempts - number
Current time	Current date.time
Source file	Source pathname
Source file size	Number of bytes; displayed only if the source file is on the local site
Destination file	Destination pathname
Source site	Source sitename
Destination site	Destination sitename
Request log file	Pathname of log file; not always displayed
Log message level	Status of messages; not always displayed



Source user	A user-id (or another name) at the source site to be associated with the transferred file; not always displayed; useful when notifying a user at a remote site about a transfer
Destination user	A user-id (or another name) at the destination site to be associated with the transferred file; not always displayed; useful when printing files at remote sites or when notifying a user at a remote site about a transfer
Options	List of active options

For example, assume your user-id is SALLY on SYSB:

```
OK, FTR PROSPECTUS IDEAS>PROJECT -DSIN_SITE SYSC
[FTR rev 1.0]
Request PROSPECTUS (9635953) submitted.
OK, FTR -DISPLAY
[FTR rev 1.0]
Request      - PROSPECTUS (9635953)
User         - SALLY
Queue        - QUEUEA
Queued       - 82-03-30.16:03:59  Status - waiting
Last attempt - 00-00-00.00:00:00  Attempts - 0
Current time - 82-03-30.16:04:08
Source file  - <DISKA>SALLY>PROSPECTUS
Destination file - IDEAS>PROJECT
Source site  - SYSB
Destination site - SYSC
Source user  - SALLY
Options :-
Binary, Copy, No Delete, No Source notify, No Destination notify.
OK,
```

### Logging Request Events

You can create an automatic log of file transfer request events by specifying the -LOG option, in the form -LOG pathname, when you submit a request. FTR will deposit logging information in pathname on the system originating the request. If pathname already exists, the logging information will be appended to the end of the file.

For example:

```
FTR SALLY>INFO WILLIAM>INFORM -DS SYSG -LOG SALLY>FTR.LOG
```



If the transfer of INFO was successful, the entries in the log file FTR.LOG would look like this:

```
15.55.13: [1.1] Request INFO (95423276) started Friday, April 23, 1982
15.55.14: [1.1] Submitting user is SALLY
15.55.14: [1.1] Local file is <DISKZ>SALLY>INFO
15.55.30: [1.1] RESULT: Transfer Terminated: Satisfactory and Complete.
15.55.30: [1.1] Request INFO (95423276) finished.
```

You can increase the degree of detail entered in your log file if you wish. Specify on the FTR command line the -MESSAGE\_LEVEL option with one of the following arguments: DETAILED, STATISTICS, TRACE. You must also specify the -LOG option. More information on the -MESSAGE\_LEVEL option appears in the PRIMENET Guide.

### Requesting Notification about Transfers

Receiving Notification Yourself: You may ask for notification about the progress of a submitted request by specifying on the FTR command line one of the following options:

<u>Option</u>	<u>Description</u>
{ -SRC_NTFY -SN }	Source notify. Use -SRC_NTFY when you are sending a file.
{ -DSTN_NTFY -DN }	Destination notify. Use -DSTN_NTFY when you are fetching a file.

You will receive messages at your terminal about your request. For example, using the last example:

```
OK, FTR PROSPECTUS IDEAS>PROJECT -DS SYSC -SRC_NTFY
[FTR rev 1.0]
Request PROSPECTUS (9635953) submitted.

***FTSERV (user 109 on SYSB) at 16:41
Request PROSPECTUS (9635953) transfer started.
OK,

***FTSERV (user 109 on SYSB) at 16:41
Request PROSPECTUS (9635953) transfer ok.
OK,
```

In this example, first message from FTSERV indicates that the file transfer has begun. The second message indicates that the transfer was completed successfully. FTSERV is the name for the FTS server. This name is set by the System Administrator and may be different on your system.



Notifying Others as Well as Yourself: You may also specify both the `-SRC_NOTIFY` and `-DSTN_NOTIFY` options together on the same command line if you wish to inform a remote source user or remote recipient as well as yourself of the progress of the transfer. In this case you must also specify on the command line at least one of the following:

<u>Option</u>	<u>Description</u>
<code>{ -DSTN_USER } user-id</code> <code>{ -DU }</code>	Required with <code>-DSTN_NOTIFY</code> when you are sending a file so that FTR will know whom to notify
<code>{ -SOURCE_USER } user-id</code> <code>{ -SU }</code>	Required with <code>-SRC_NOTIFY</code> when you are fetching a file so that FTR will know whom to notify

For example, if your command line in the previous example had been:

OK, FTR PROSPECTUS JOHN>PROJECT -DS SYSC -SN -DN -DSTN USER JOHN

the messages listed above would have been sent both to you and to JOHN on SYSC.

#### Note

Because of the way `-SRC_NOTIFY` and `-DSTN_NOTIFY` work with the PRIMOS MESSAGE facility, you may not always receive all of your FTS notifications. We recommend that you use the `-LOG` option instead of, or in addition to, `-SRC_NOTIFY` and `-DSTN_NOTIFY`.

#### Cancelling Requests

If you have submitted a request that is currently waiting in a queue to be transferred, you may cancel the request with the command:

```
FTR -CANCEL { request-name }
             { request-number }
```

For example, if the request you wished to cancel was named NEWS and the request-number was 5684210, either of the following commands would cancel the request:

```
FTR -CANCEL NEWS
FTR -CANCEL 5684210
```

You will receive the following message:

Request NEWS (5684210) cancelled.



You cannot cancel requests that are in the process of being transferred.

### FTR's Help Facility

You can request a brief summary of FTR options at the terminal by specifying simply:

```
FTR
```

### Requests on Hold

Under certain circumstances (such as an invalid directory password), FTR will automatically put the request "on hold". If you should receive a message to this effect, or see in an FTR -DISPLAY screen that your request is on hold, you may wish to do one of the following:

- Give the command:

```
FTR -CANCEL { request-name }
              { request-number }
```

This command will cancel the request, and you can resubmit it.

- Give the command:

```
FTR -RELEASE { request-name }
              { request-number }
```

This command will instruct FTR to try the transfer again.

- Try to determine what may have caused the failure, and resubmit a corrected request.

Both you and the operator can hold a file intentionally with the -HOLD option. More information on this feature is in the PRIMENET Guide.

### Other Options

The FTR command admits other options that allow you to modify, abort, and otherwise control your requests. Full information on these options (as well as on the -HOLD and -RELEASE options) appears in the PRIMENET Guide.



USING REMOTE IDSIntroduction

Normally, in a Prime network, user ids on any one system are recognized by the other (remote) system or systems. Work in directories on one remote system is processed simply by giving commands in the customary way that you work in directories on your own local system. What actually happens is that a special kind of phantom called a "slave" uses your user id, and does the work on the remote system for you. This process is virtually invisible to the user.

Sometimes, for security reasons, your System Administrator will prevent certain remote systems from recognizing your normal user id. If your user id is not recognized by the remote system, the slave cannot do your work with this id on that system. You must establish another id, called a remote id, that the remote system does recognize. The slave will use this remote id. You must add the remote id before you try to access the remote system.

You establish a remote id on your own system (not on a remote system), just as your user id is established on your own system. The id is called a "remote id" because it allows you to use otherwise inaccessible remote systems.

Note

You might also wish to establish a remote id that a slave will use on remote systems where your normal user id is recognized. For example, a remote id may allow you certain ACL rights that your own user id does not allow.

Establishing Remote Ids

You can establish a remote id with the `ADD_REMOTE_ID` command, which has the following format:

```
ADD_REMOTE_ID remote-id [password] -ON systemname [-PROJECT project-id]
```

Abbreviation for `ADD_REMOTE_ID`: `ARID`

The arguments and options for this command are the same as those for the `LOGIN` command. (The `LOGIN` command was explained initially in Chapter 3 and reviewed earlier in this chapter in the section "Remote Login".)

remote-id is the user id that the slave will use for you on the remote system specified by systemname. In order to be useful, remote-id must already exist on the systemname. That is, it must have been defined as a valid user id by an administrator on systemname before your slave can actually use it on the remote system. You must also supply any



password or project-id required for access to the remote system. If the remote id does not exist on the remote system, or if any required password or project id is missing or incorrect, a subsequent attempt to access the remote system will fail.

You may have remote ids for up to 16 different systems simultaneously, but only one for any given remote system. For example, if you add the remote id JINKS on SYSA and then add the remote id LYNX on SYSA, LYNX will replace JINKS. All remote ids are removed when you log out.

Establishing a remote id allows you to access a remote system directory with ordinary PRIMOS commands (such as ATTACH and COPY). These commands would not otherwise work. Using remote ids makes the other possible but less flexible ways of accessing the remote system (such as the use of NETLINK or FTR) unnecessary.

For example, assume your user id is JAMES on SYSP:

```
ADD_REMOTE_ID JAKE PASS -CN SYSK -PROJECT SALES
```

JAKE now exists with the password PASS as a remote id for accessing SYSK. Jake is associated with a project named SALES. JAKE must already have been defined as a valid user id with the password PASS on SYSK in order for you to do work there.

### Examining Your Remote Ids

With the LIST\_REMOTE\_ID command, you can examine the existing remote id's you have established. The format is:

```
LIST_REMOTE_ID [-CN systemname]
```

Abbreviation for LIST\_REMOTE\_ID: LRID.

If you give the -CN option, only the remote id for systemname will be listed; if you omit the -CN option, all of your remote id's will be displayed. Passwords are never displayed. For example:

```
OK, LIST_REMOTE_ID
System  User id                Project id
-----  -
SYSB    JIM
SYSK    JAKE                      SALES
SYSM    JODY
OK,
```



# 15

## Subroutine Libraries

### INTRODUCTION

This chapter lists the subroutines of primary interest in applications and available through:

- The Applications Libraries: VAPPLB (V-mode) and APPLIB (R-mode)
- The Search and Sort Libraries: VSRTLI and VMSORT (V-mode), SRTLIB and MSORTS (R-mode)
- The System Libraries: PF\*INLB (V-mode) and F\*INLIB (R-mode)
- The Condition Mechanism

It is meant solely as a checklist, to tell you what subroutines are available in these libraries. The Subroutines Reference Guide tells you how to use them. Thus, if you wanted to know whether a certain sort routine was available, you would look for it here. Having found it, you would consult the Subroutines Reference Guide for full details on how to call and use it.

Other subroutine libraries exist for individual products. These are discussed in the appropriate manual for the product. For example, each language library is discussed in the relevant language reference book; the network library (VNETLB) is discussed in the PRIMENET Guide.



Note

At Rev. 19.0 the names of all of the object libraries (libraries loaded with programs) now end in .BIN (for example: VSRTLI.BIN). With one exception, this suffix need not be specified when you load the libraries. (If you use the LIBEDB utility, described in the Subroutines Reference Guide, you must give the .BIN suffix.) In this chapter the .BIN suffix has been omitted.

Keys and Error Codes

All keys and error codes are specified in symbolic, rather than numeric form. These symbolic names are defined as PARAMETERS in insert files in a UFD on the master disk called SYSCOM. The names of these files for their appropriate languages are given in Table 15-1. All keys are INTEGER\*2 (that is, 16-bit integers).

Table 15-1  
Files for Keys and Error Codes

Language	Keys Insert File	Errors Insert File
FORTRAN IV	KEYS.INS.FTN	ERRD.INS.FTN
FORTRAN 77	KEYS.INS.FTN	ERRD.INS.FTN
Pascal	KEYS.INS.P	ERRD.INS.PASCAL
PL/I Subset G	KEYS.INS.PL1	ERRD.INS.PL1
PMA	KEYS.INS.PMA	ERRD.INS.PMA
COBOL	not applicable	not applicable
BASIC/VM	not applicable	not applicable

The Subroutines Reference Guide explains how to use each file in each language, and how to make substitutions in BASIC and COBOL.



Error Handling

Many subroutines have an argument called CODE, which returns a code according to the success or failure of the subroutine call. The meaning of these codes is given in Appendix D of the Subroutines Reference Guide.

Errors occurring from a subroutine call cause a non-zero value of the argument CODE to be returned. Users should always test CODE after a call for non-zero values to be certain no errors are missed. Error printing and control may be performed by the ERRPR\$ subroutine from the System library.

APPLICATIONS LIBRARIES

The applications libraries provide programmers with easy-to-use functions and service routines that fall between very high-level constructs and very low-level systems routines. The applications libraries are located in the UFD LIB in the files APPLIB (R-mode programs) and VAPPLB (V-mode programs). All routines in VAPPLB are pure procedure; that is, they contain no data mixed with the code and are therefore sharable. These routines may be loaded into the shared portion of a shared procedure. The appropriate applications library (VAPPLB or APPLIB) should be loaded before loading the FORTRAN library.

Programs using the applications libraries subroutines should define the values of the keys used in these routines. This definition is performed by placing the instruction INSERT SYSCOM>A\$KEYS.INS.language in each module which uses any of these subroutines. These files are not available for BASIC/VM, COBOL, or PMA; programs in these languages must use the numeric values of these keys.

The applications routines may be used as functions or as subroutine calls as desired. The function usage gives additional information. The type of value of the function (LOGICAL, INTEGER, and so on) is specified for each function.

Applications Libraries Subroutines Listed by Function

The applications libraries subroutines may be grouped by their functions:

File System: TEMP\$A, OPEN\$A, OPNP\$A, OPNV\$A, OPVP\$A, CLOS\$A, RWND\$A, GEND\$A, TRNC\$A, DELE\$A, EXST\$A, UNIT\$A, RPOS\$A, POSN\$A, TSCN\$A.

String Manipulation: FILL\$A, NLEN\$A, MCHR\$A, GCHR\$A, TREE\$A, TYPE\$A, MSTR\$A, MSUB\$A, CSTR\$A, CSUB\$A, LSTR\$A, LSUB\$A, JSTR\$A, FSUB\$A, RSTR\$A, RSUB\$A, SSTR\$A, SSUB\$A



User Query: YSNO\$A, RNAME\$A, RNUM\$A

System Information: TIME\$A, CTIM\$A, DTIM\$A, DATE\$A, EDAT\$A, DOFY\$A

Conversions: ENCD\$A, CNVA\$A, CNVB\$A, CASE\$A, FDAT\$A, FEDT\$A, FTIM\$A

Mathematical Routines: RNDI\$A, RAND\$A

Parsing: CMDL\$A

### Descriptions of Applications Libraries Subroutines

Brief descriptions of the subroutines in the applications libraries follow in alphabetical order. The header on the right is the type of function. Note that LOGICAL, .TRUE., AND .FALSE. are FORTRAN values, not Boolean values. INTEGER\*2 is a 16-bit integer; INTEGER\*4 is a 32-bit integer. REAL\*4 and REAL\*8 are 32-bit and 64-bit floating-point numbers, respectively.

CASE\$A

LOGICAL

Converts a character string from uppercase to lowercase or vice versa and returns .TRUE. if operation succeeds.

CLOS\$A

LOGICAL

Attempts to close a file by the file unit number on which it was opened. Reports on success or failure of attempt. (A file unit is a logical unit that PRIMOS associates with an open file. File unit numbers range from 1 to 126. Further information on file units appears in the Subroutines Reference Guide.)

CMDL\$A

LOGICAL

Parses a PRIMOS-like command line and returns information for each keyword (and optional argument) entry in the line (one entry is returned per call).

CNVA\$A

LOGICAL

Converts an ASCII digit string to a numerical value (INTEGER\*4) for binary, octal, decimal, and hexadecimal numbers. Reports whether the conversion was made successfully or not.



CNVB\$A

INTEGER\*2

Converts a number (INTEGER\*4) to an ASCII digit string for binary, decimal, octal, and hexadecimal numbers. The function value is the number of digits in the string (or 0 if the conversion is unsuccessful).

CSTR\$A

LOGICAL

Compares two character strings for equality and returns .TRUE. as the function value if they are equal.

CSUB\$A

LOGICAL

Compares two substrings of character strings for equality and returns .TRUE. as the function value if they are equal.

CTIM\$A

REAL\*4 or REAL\*8

Returns the CPU time since login in centiseconds (argument returned) and in seconds (function value). (One centisecond = .01 second.)

DATE\$A

REAL\*8

Returns the system date as DAY, MON DD 19YR (argument returned) and as MM/DD/YY (function value).

DELE\$A

LOGICAL

Attempts to delete a file specified by the filename. If successful the function is .TRUE., otherwise .FALSE..

DOFY\$A

REAL\*8

Returns the day of the year as a 3-digit number (argument returned) and as YY.DDD (function value). The latter is suitable for printing in FORTRAN format F6.3.

DTIM\$A

REAL\*8

Returns disk time since login in centiseconds (argument returned) and in seconds (function value).



EDAT\$A REAL\*8

Returns the date as DAY, DD MON YYYY (argument returned) and as DD.MM.YY (function value). This is the European/military format.

ENCD\$A LOGICAL

Encodes a value in FORTRAN floating-point print format (Fw.d) and reports whether the encoding was successful or not.

EXST\$A LOGICAL

Checks for the existence of a file specified by name and reports whether the file exists or not.

FDAT\$A REAL\*8

Converts the date-last-modified (DATMOD) field of a directory entry to DAY, MON DD YEAR (argument returned) and MM/DD/YY (function value). For use only with RDEN\$\$.

FEDT\$A REAL\*8

Converts the date-last-modified (DATMOD) field of a directory entry to DAY, MON DD YEAR (argument returned) and MM.DD.YY (function value). For use only with RDEN\$.

FILL\$A INTEGER\*2

Fills a character string with a specified ASCII character.

FSUB\$A LOGICAL

Fills a character substring with a specified character and returns .TRUE. if successful.

FTIM\$A REAL\*4 or REAL\*8

Converts the time-last-modified (TIMMOD) field of a directory entry to HH:MM:SS (argument returned) and decimal hours (function value).

GCHR\$A INTEGER\*2 or INTEGER\*4

Accesses a character in a specified character position. The function value is the character in FORTRAN A1 format, right-padded with blanks.



GEND\$A

LOGICAL

Positions a file pointer opened on a specified file unit to the end of file. The function value tells whether the positioning was successful or not.

JSTR\$A

LOGICAL

Justifies left or right, or centers a string and reports whether the operation is successful.

LSTR\$A

LOGICAL

Locates a string within another string. The function value reports on whether the substring was found or not.

LSUB\$A

LOGICAL

Locates one substring within another substring. The function value reports on whether the substring was found or not.

MCHR\$A

INTEGER

Replaces a character in one array with a specified character from another. The function value is the character moved in FORTRAN A1 FORTRAN, right-padded with blanks.

MSTR\$A

INTEGER

Moves one string to another string. The function value is equal to the number of characters moved.

MSUB\$A

INTEGER

Moves a substring in one string into a substring in another string. The function value is equal to the number of characters moved.

NLEN\$A

INTEGER\*2

Returns the length (not including trailing blank) of a string in a buffer.



## OPEN\$A

LOGICAL

Opens a file on a user-specified or system-specified file unit. The function value reports whether the operation was successful or not.

## OPNP\$A

LOGICAL

Gets a filename from the user terminal and opens that file on a specified file unit. The function value reports whether the operation was successful or not.

## OPNV\$A

LOGICAL

Opens a file on a user-specified or system-specified file unit and verifies the operation. If the file is in use, the operations are retried. The function value reports on the ultimate success of the operations.

## OPVP\$A

LOGICAL

Gets a file name from the user terminal and opens that file on a specified file unit. The operations are verified. If the file is in use, the operations are tried again. The function value reports on the ultimate success of the operations.

## POSN\$A

LOGICAL

Positions the pointer in the file open on a specified file unit. The function value reports on the success of the operation.

## RAND\$A

REAL\*4 or REAL\*8

Updates the seed of a random number generator. The old seed is passed and a new seed returned. The function value is a uniform random number between 0.0 and 1.0.

## RNV\$A

LOGICAL

Prints a prompt message at the terminal and accepts a name from the terminal. The function value reports on the validity of the name.

## RNDI\$A

REAL\*4 or REAL\*8

Generates the initializing seed for a random number generator. The information returned is time of day in centiseconds (argument returned) and in seconds (function value).



## RNUM\$A

LOGICAL

Prints a prompt message at the terminal and accepts a number (octal, decimal, or hexadecimal) string from the terminal. If successful, the value is returned in one of the subroutine arguments and the function value is .TRUE..

## RPOS\$A

LOGICAL

Returns the current absolute position of the pointer in the file opened on a specified file unit. The function value reports on the success of the operation.

## RSTR\$A

LOGICAL

Rotates a character string left or right, after truncating the string to its operational length (ignoring trailing blanks).

## RSUB\$A

LOGICAL

Rotates a character substring left or right. Affects only the characters of the substring contained in the specified string.

## RWND\$A

LOGICAL

Rewinds the file opened on the specified file unit. The function value reports on the success of the operation.

## SSTR\$A

LOGICAL

Shifts a character string left or right. Shifts the specified number of characters and pads the vacated positions with the specified fill character. Does not include trailing blanks in the shift.

## SSUB\$A

LOGICAL

Shifts a character substring left or right. Shifts the specified number of characters and pads the vacated positions with the specified fill character. Includes trailing blanks in the shift.

## TEMP\$A

LOGICAL

Opens a temporary file with a unique name in the current UFD for reading and writing on a user-specified or system-specified file unit. The name is returned as an argument in the subroutine call. The function value reports on the success of the operation.



TIME\$A

REAL\*8

Returns the time of day as HR:MN:SC (argument returned) and in decimal hours (function value).

TREE\$A

LOGICAL

Scans a string to check whether it is a valid pathname and, if so, locates the final part (filename) of the name in the string. The function value reports whether the test is successful or not.

TRNC\$A

LOGICAL

Truncates the file opened on a specified file unit. The function value reports on the success of the operation.

TSCN\$A

LOGICAL

Scans the file system tree structure (starting with the home directory) to read UFDs and segment directory entries. Each call returns the next file on the current level or the first file on the next lower level. The function value is .TRUE. until an error occurs or an end of file is reached.

TYPE\$A

LOGICAL

Tests a character string to see whether it can be interpreted as a number (binary octal, decimal, or hexadecimal) or a name. The function value reports whether the string meets the specified criterion.

UNIT\$A

LOGICAL

Tests whether any file is open on a specified file unit. The function value reports whether the unit is in use or not.

YSNO\$A

LOGICAL

Prints a question at the user terminal which can be answered YES (or OK) or NO. The function value is .TRUE. for YES (or OK) and .FALSE. for NO. Any other answer causes the question to be repeated.

#### SORT AND SEARCH LIBRARIES

There are two classes of sorting subroutines available: disk sorts and in-memory sorts. Disk sorts use the mass storage devices (disks) for working space, while the in-memory sorts put working information in the



user's address space. For complete details on the use of these subroutines, see the Subroutine Reference Guide.

### Disk Sorts

Disk sort subroutines are in the VSRTLI (V-mode) and SRTLIB (R-mode) libraries.

VSRTLI: VSRTLI contains the following:

- ASCSRT and ASCS\$\$ sort or merge ASCII or binary files on any of the 13 supported key types. ASCSRT is available only in V-mode.
- SUBSRT sorts a single input file on ASCII keys. It has a simpler calling sequence than ASCS\$\$.
- SRTF\$\$ sorts from one to twenty input files into a single output file. It allows specification of both input and output file types.
- MRG1\$\$ merges from one to eleven input files into a single output file. It allows specification of both input and output file types.
- MRG2\$\$ returns the next merge record.
- MRG3\$\$ closes the merge input files.

A set of cooperating sort routines can be used together in a program to provide not only sorting but also input and output procedures for preprocessing and postprocessing. These are:

<u>Name</u>	<u>Activity</u>
SETU\$\$	Prepares sort table and buffers
RLSE\$\$	Gets input records from procedure or file
QMBN\$\$	Sorts
RTRN\$\$	Sends sorted records to file or output procedure
CLNU\$\$	Closes all sort files

The functioning of these routines, and the tradeoffs involved, are discussed in the Subroutines Reference Guide.



SRTLIB: SRTLIB contains the following:

- ASCS\$\$ sorts on ASCII (upper and lower case) or binary keys. It can also merge up to ten files.
- SUBSRT sorts a single input file on ASCII keys. It has a simpler calling sequence than ASCS\$.

### In-memory Sorts and Binary Search

The subroutines listed here are contained in the library MSORTS (R-mode) and VMSORTS (V-mode) in the UFD LIB. A complete discussion of these subroutines will be found in the Subroutines Reference Guide.

For a complete discussion of these types of sorts, see Donald Knuth, The Art of Computer Programming, vol. 3, Reading, Mass.: Addison-Wesley, 1973.

Table 15-2 lists the characteristics of these sorts.

Table 15-2  
Sort Characteristics

Sort	Approximate Relative Running Time		Comments
	Average	Maximum	
BINSRCH	$8.5 \cdot \ln(N) - 6$	$8.5(\ln(n)) + 12$	searches or builds a table whose keys are in order
BUBBLE	$N^2$	-	only good for very small N
HEAP	$23N \cdot \ln(N)$	$26N \cdot \ln(N)$	inefficient for $N < 2000$
INSERT	$N^2$	-	small N; very good on nearly ordered tables
QUICK	$12N \cdot \ln(N)$	$N^2$	fastest, but very slow on nearly ordered tables
RADXEX	$14 \cdot N \cdot \ln(N)$	-	very fast for large N, as long as no equal keys are present
SHELL	$N^{1.25}$	$N^{1.5}$	good for $N < 2000$
"N" is the number of entries in the table.			



These routines, except for RADSEX, all sort the table in increasing order with the key treated as a single, signed multiword integer.

RADSEX, however, treats the key as a single, unsigned multiword (or partial word) integer. For example: If the keys were 5, -1, 10, -3, RADSEX would sort them to: 5, 10, -3, -1. The other routines would sort them to: -3, -1, 5, 10.

### SYSTEM LIBRARIES

A list of most system subroutines and groups of subroutines with a brief description of their functions, is given below. These subroutines are used mainly by PRIMOS, although some are useful at the applications level. Complete details are found in the Subroutines Reference Guide.

Following the list is a full description of a sample subroutine and an example of how to call it from a user program.

<u>Subroutine(s)</u>	<u>Description</u>
ACLs Subroutines	Manipulate the access control list protection system. (The ACL subroutines are listed and discussed in the <u>Subroutines Reference Guide</u> .)
APSFx\$	Appends a suffix to a pathname.
ATTACH Subroutines	Attach to a directory. At Rev. 19, six new ATTACH subroutines (AT\$, AT\$ABS, AT\$ANY, AT\$HOM, AT\$OR, and AT\$REL) may be used in place of the older subroutine ATCH\$.
BREAK\$	Inhibits or enables CONTROL-P.
CLIN\$	Gets one character from command file or terminal.
CNAM\$\$	Changes the name of a file system object.
CL\$GET	Reads a line of text from a command file or terminal.
CL\$PIX	Parses a command line.
COMI\$\$	Switches command input stream from terminal to command file and vice-versa.
COMO\$\$	Switches output stream from terminal to file and vice-versa.



<u>Subroutine(s)</u>	<u>Description</u>
CREA\$\$	Creates a sub-UFD in the current UFD.
Date Subroutines	At Rev. 19, five new date-conversion subroutines have been added (CV\$FDA, CV\$FDV, CV\$DQS, CV\$DTB, and DATE\$).
DUPLX\$	Returns terminal configuration word.
ERKL\$\$	Reads or sets the erase and kill characters.
ERRPR\$	Interprets a return code.
EXIT	Returns to PRIMOS.
FNCHK\$	Checks a filename for valid format.
FORCEW	Writes immediately to the disk all modified records of the file currently open on file unit.
GCHAR	Gets a character from an array.
GPAS\$\$	Returns passwords of sub-UFD in the current UFD.
GV\$GET	Retrieves the value of a global variable.
GV\$SET	Sets the value of a global variable.
GPATH\$	Obtains a fully qualified pathname for an open file unit.
IDCHK\$	Checks an id for valid format.
LOGO\$\$	Logs out a user or process.
LON\$CN	Enables or disables logout notification.
LON\$R	Retrieves phantom logout notification information.
Message Subroutines	At Rev. 19, four new subroutines (MGSET\$, MSG\$ST, SMSG\$, RMSGD\$) have been added to support the inter-user message facility.
NAMEQ\$	Compares filenames for equivalence.
PHNIM\$	Starts a phantom (same login name only).
PRWF\$\$	Reads, writes, and positions pointer in a SAM or DAM file.



<u>Subroutine(s)</u>	<u>Description</u>
PWCHK\$	Checks a login password for valid format.
Q\$READ	Reads quota information.
Q\$SET	Sets maximum quota.
RECYCL	Passes control to next user.
RDEN\$\$	Reads entry in UFD.
RDLIN\$	Reads line of characters from compressed or uncompressed ASCII disk file.
REST\$\$	Restores an R-mode executable object file to user memory from a disk file.
RESU\$\$	Restores an R-mode executable object file from a file, sets initial values, and begins execution.
SATR\$\$	Sets attributes (protection, date, time, etc.) in a UFD entry.
SAVE\$\$	Saves an R-mode executable object file in user memory by writing it into a disk file.
SCHAR	Stores a character in an array.
SGDR\$\$	Positions and reads segment directory entries.
SPAS\$\$	Sets the passwords in the current UFD.
SPOOL\$	Inserts a file in spooler queue.
SRCH\$\$	Opens or closes a file.
SRSFX\$	Searches for a file with one of a list of suffixes.
TEXTOS	Checks the validity of a filename (obsolete for PL1G and Pascal programmers; replaced by FNCHK\$).
TIMDAT	Returns system and user information.
TNCHK\$	Checks a pathname for valid format.
TSRC\$\$	Opens or closes a file anywhere in the PRIMOS file structure.



<u>Subroutine(s)</u>	<u>Description</u>
USER\$	Returns process number and user count.
UTYPE\$	Returns type of current process.
WTLIN\$	Writes a line of ASCII characters to a disk file in compressed format.

#### Sample Full Description of a System Subroutine - CNAM\$\$

The following information describes the calling sequence and parameters needed to use the subroutine CNAM\$\$ . CNAM\$\$ changes the name of a file system object.

The calling sequence is:

CALL CNAM\$\$ (old-name,old-name-length,new-name,new-name-length, code)

<u>Argument</u>	<u>Description</u>
<u>old-name</u>	Name of file system object to be changed.
<u>old-name-length</u>	Number of characters in <u>old-name</u> .
<u>new-name</u>	Name to be changed to.
<u>new-name-length</u>	Number of characters in <u>new-name</u> .
<u>code</u>	Returns integer-valued error code.

#### Note

CNAM\$\$ requires delete (D) and add (A) access in the current directory in ACL systems, or owner rights in the current directory in password systems.

The names of the MFD, BOOT, or BADSPT may not be changed.



Sample COBOL Program

The following COBOL program illustrates how to call a subroutine from a user program. The program uses the CNAME\$\$ subroutine described fully above.

OK, SLIST CHANGE.COBOL

IDENTIFICATION DIVISION.

PROGRAM-ID. CHANGE.

ENVIRONMENT DIVISION.

DATA DIVISION.

WORKING-STORAGE SECTION.

01 OLD-NAME PIC X(20).

01 OLD-LENGTH COMP VALUE 20.

01 NEW-NAME PIC X(20).

01 NEW-LENGTH COMP VALUE 20.

01 ERR-CODE COMP.

01 RETRY PIC X(3) VALUE 'YES'.

77 FLAG PIC X(7) VALUE 'NOTDONE'.

PROCEDURE DIVISION.

000-BEGIN.

IF RETRY = 'YES' PERFORM 010-PROCESS UNTIL FLAG = 'DONE',

ELSE DISPLAY 'END OF RUN'.

STOP RUN.

\*

010-PROCESS.

DISPLAY 'ENTER NAME OF FILE TO BE CHANGED.'.

ACCEPT OLD-NAME.

DISPLAY 'ENTER NEW NAME.'.

ACCEPT NEW-NAME.

CALL 'CNAM\$\$' USING OLD-NAME, OLD-LENGTH, NEW-NAME,  
NEW-LENGTH, ERR-CODE.

IF ERR-CODE NOT EQUAL 0 PERFORM 050-ERROR,

ELSE DISPLAY 'CHANGE WAS SUCCESSFUL',

MOVE 'DONE' TO FLAG, STOP RUN.

EXIT.

\*

050-ERROR.

DISPLAY 'UNSUCCESSFUL — ERROR CODE IS ', ERR-CODE.

DISPLAY 'DO YOU WANT TO TRY AGAIN — YES OR NO.'.

ACCEPT RETRY.

IF RETRY = 'NO' MOVE 'DONE' TO FLAG.

EXIT.



If the file `OLDFILE` exists, the program may be compiled, loaded, and run with the following dialog.

OK, COBOL CHANGE

Phase I  
Phase II  
Phase III  
Phase IV  
Phase V  
Phase VI

No Errors, No Warnings, Prime V-Mode COBOL, Rev 19.0 <CHANGE>

OK, SEG -LOAD

[SEG rev 19.1.0]

\$ LO CHANGE

\$ LI VCOBIB

\$ LI

LOAD COMPLETE

\$ EXEC

ENTER NAME OF FILE TO BE CHANGED.

OLDFILE

ENTER NEW NAME.

NEWFILE

CHANGE WAS SUCCESSFUL

If the file `OLDFILE` does not exist, running the program produces the following result.

OK, SEG CHANGE

ENTER NAME OF FILE TO BE CHANGED.

OLDFILE

ENTER NEW NAME.

NEWFILE

UNSUCCESSFUL — ERROR CODE IS 00015+

DO YOU WANT TO TRY AGAIN — YES OR NO.

NO

OK,

#### CONDITION MECHANISM SUBROUTINES

The condition mechanism is activated when a program encounters unexpected occurrences, such as end of file, illegal address, an attempt to divide by zero, or use of the `BREAK` key from a terminal.

The condition mechanism's goal is either to repair the problem and restart the program, or to terminate the program in an orderly manner. To achieve this goal, the condition mechanism activates diagnostic or remedial blocks of code called on-units.



The user-level subroutines for the condition mechanism are listed in Table 15-3.

Table 15-3  
Condition Mechanism Subroutines

Subroutines (by Programming Language)				
Action	FTN	F77	PL1G	PMA
Call more on-units	CNSIG\$	CNSIG\$	CNSIG\$	CNSIG\$
Create an on-unit	MKON\$F	MKON\$F	MKON\$P	MKONU\$
Signal a condition	SGNL\$F	SGNL\$F	SIGNL\$	SIGNL\$
Cancel (revert) an on-unit	RVON\$F	RVON\$F	RVONU\$	RVONU\$
Nonlocal GOTO	PL1\$NL	PL1\$NL	*	PL1\$NL
Make PL/I-compatible label	MKLB\$F	MKLB\$F	*	MKLB\$F
*Supported directly by the programming language.				

More detailed information on the Condition Mechanism and its associated subroutines appears in Chapter 20 of this manual and in the Subroutines Reference Guide.



Report of the Committee on the Administration of the Court

Table 1: Summary of the Court's Work

Summary of the Court's Work				
Case No.	Case Name	Case Type	Case Status	Case Outcome
10001	John Doe vs. Jane Smith	Civil	Settled	Settlement reached
10002	State vs. John Doe	Criminal	Verdict	Guilty
10003	State vs. Jane Smith	Criminal	Verdict	Not Guilty
10004	State vs. John Doe	Criminal	Verdict	Guilty
10005	State vs. Jane Smith	Criminal	Verdict	Not Guilty
10006	State vs. John Doe	Criminal	Verdict	Guilty
10007	State vs. Jane Smith	Criminal	Verdict	Not Guilty
10008	State vs. John Doe	Criminal	Verdict	Guilty
10009	State vs. Jane Smith	Criminal	Verdict	Not Guilty
10010	State vs. John Doe	Criminal	Verdict	Guilty

The Court has heard 10 cases in the past year. The majority of cases are criminal cases. The Court has reached a verdict in all cases.



**PART IV**

**The Command Environment**



PART IV

The General Principles



# 16

## Protecting Your Files and Directories

### INTRODUCTION

You may wish to secure your files and directories against unauthorized users. PRIMOS provides a protection mechanism for this purpose known as access control lists (ACLs). Access control lists specify who may have access to a file system object and exactly what kind of rights each user has.

The ACL system is extremely flexible. Various access rights may be granted or denied to a single user or to a group of users. An ACL can protect a single object or a set of objects.

Protection may be very loose; for example, all users may be granted all rights. On the other hand, protection may be very secure; no one except the "owner" of an object may be allowed any access at all.

Moreover, if you do not wish to use the system actively, you can establish one ACL (or have your System or Project Administrator establish one for you) that will automatically protect all your work in a given directory and its subtree. This is called default protection.

### Note

Directory passwords and owner/non-owner access rights provide an alternative to access control lists as a protection system for PRIMOS file system objects. The directory password system and its related commands (PASSWD and PROTECT) are discussed in



Appendix F. Appendix F also discusses how to convert password-protected directories to ACL-protected directories, and vice-versa.

This chapter will explain the ACL system, including:

- What ACLs are and what they look like
- Types of access rights and types of users
- What access categories are
- Types of protection (specific ACLs, access categories, and default protection)
- Who may distribute rights
- Commands for using ACLs and access categories
- Tips on setting access rights effectively

#### WHAT ARE ACCESS CONTROL LISTS (ACLs)?

An access control list (ACL) is a list of users and the access privileges granted to each. When an ACL is associated with a file, a directory, or a segment directory, it becomes the protective mechanism for that object.

A simple ACL could look like this:

```
BILL:      ALL
$REST:     NONE
```

In this example, user BILL has all rights to the object that this ACL protects. \$REST, a special designation indicating "everybody else", has no rights.

When any user gives a command concerning an ACL-protected file or directory, PRIMOS checks the user id against the ACL associated with the file or directory. If the user has the appropriate access rights, the command is executed; if not, the command is not executed, and PRIMOS returns the message "Insufficient access rights" or "No Information".

#### Types of Access Rights

The access rights or privileges that ACLs grant to users are summarized in Table 16-1.



Table 16-1  
ACL Access Rights

Symbol	Right	Applies To	Meaning
R	Read	Files	File may be read.
W	Write	Files	File may be modified.
U	Use	Directories	User may attach to directories.
L	List	Directories	Directory contents may be listed.
A	Add	Directories	Directory entries may be added.
D	Delete	Directories	Directory entries may be deleted.
P	Protect	Directories	Access rights may be changed.
ALL		Files and Directories	All of the above rights.
NONE		Files and Directories	No access allowed.



Definitions, uses, and combinations of access rights are discussed more fully at the end of this chapter in the section TIPS ON SETTING ACCESS RIGHTS.

### Types of Users

Access rights are granted to users. The designation (or identifier) for users in the ACL may be any of three types:

- A user id (for example, "JONES").
- A group name. All group names begin with a period (for example, ".COMMITTEE"). A number of user id's are associated with a group, and each of them has the access rights granted to the group name. For example ".COMMITTEE" might be composed of user id's "JANE", "FRED", and "BARBARA". An individual user may be a member of up to 32 groups. Groups are established by the System Administrator or by a Project Administrator.
- The special identifier \$REST. \$REST specifies the rights granted to all users not otherwise identified in the ACL (that is, "everybody else").

### What an ACL Looks Like

An ACL has a specific format. The identifier (user id, group name, or \$REST) is separated from the rights associated with it by a colon. On the terminal, the pairs of identifiers and rights are arranged in columns.

For example, consider the following ACL:

```
CAROL:      ALL
.COMMITTEE: RW
$REST:      R
```

In this example, an ACL gives CAROL all access rights (protect, delete, add, list, use, read, and write). Everyone in the .COMMITTEE group has read and write access, and everyone else (that is, \$REST) has read access.

### Overlapping Access Rights

A user who is in more than one group is granted the sum (logical union) of all access modes for those groups. If a user has access rights from both a user id and a group name, the user id takes precedence and he or she receives his or her user id rights only.



Consider the following example. The groups .ANIMALS and .RODENTS are comprised of the following users:

.ANIMALS

SQUIRREL  
ROBIN  
MOUSE  
DOG

.RODENTS

SQUIRREL  
MOUSE

The directory FEEDER is protected by an ACL that contains the following entries:

DOG: NONE  
.ANIMALS: DLU  
.RODENTS: ALU  
\$REST: LU

ROBIN has delete, list, and use access to FEEDER as a member of the .ANIMALS group. SQUIRREL and MOUSE have delete, add, list, and use access as the sum of rights granted to the .RODENTS and .ANIMALS groups. DOG has no access; although DOG is in the .ANIMALS group, user id rights (here NONE) take precedence over group rights. WORM, another user, has list and use access, since he is covered by the \$REST identifier.

### SPECIFIC ACLS AND ACCESS CATEGORIES

An ACL may exist in either of the following two forms:

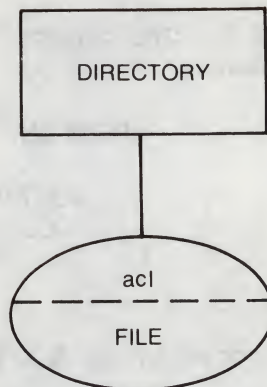
- As an unnamed ACL, that is, an unnamed attribute of a file system object, called a specific ACL
- As a named ACL, that is, a named file system object, called an access category

These definitions are expanded below.

#### Specific ACLs (Unnamed ACLs)

The list may exist as an attribute of a specific file system object, and not as a named file system object in itself. The list will not appear in a directory listing and is called a specific ACL. It is linked to the single object it protects and has no separate existence of its own. Figure 16-1 illustrates an access control list protecting a file in a directory.



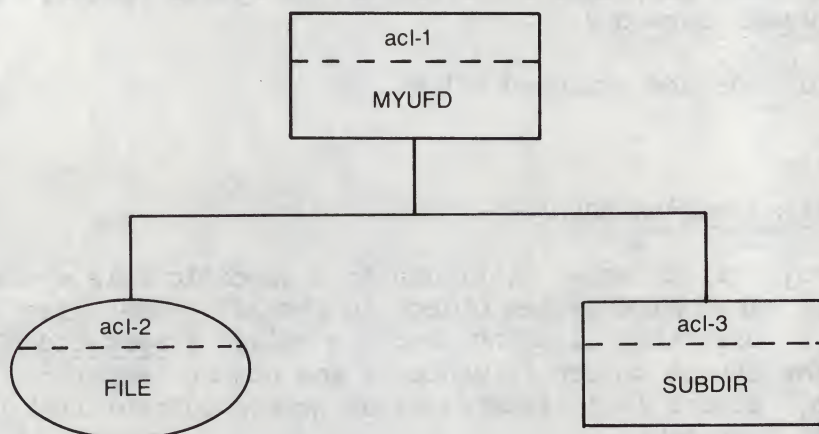


An ACL Protecting a File  
Figure 16-1

You may set a specific ACL on a file, segment directory, or directory. By doing so, you may define precisely who may access the object and exactly what rights each user has. If you set a specific ACL on an object and later delete the object, the ACL (as an attribute of the object) is deleted along with it.

If you set an ACL on a directory, it automatically protects all objects within that directory (and within its subtree) that are not otherwise protected. This is explained in more detail in the discussion DEFAULT PROTECTION, below.

Figure 16-2 illustrates three specific ACLs in a directory tree. acl-1 protects the ufd MYUFD. acl-2 protects the file FILE. acl-3 protects the subdirectory SUBDIR. Each one has been set individually.



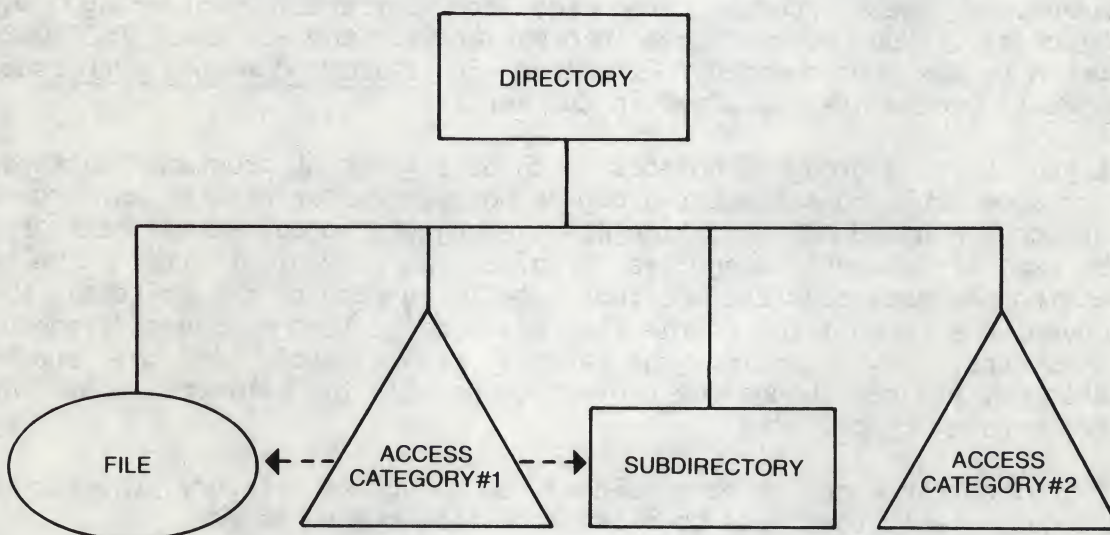
Specific ACLs Protecting Objects  
Figure 16-2



Access Categories (Named ACLs)

An access control list may also exist as a named file system object (such as "GUARD.ACAT") that resides within a directory at any level. In this form the list is called an access category. It differs from a specific ACL in that it exists as a separate, named file system object. Moreover, an access category may exist without being linked to any object. It may protect any number of files, directories, and segment directories; or, it may protect none. An access category must reside in the directory containing the object(s) it explicitly protects.

Figure 16-3 illustrates two access categories, a file, and a subdirectory in a directory. Access category #1 is protecting the file and the subdirectory. Access category #2 is not currently protecting anything.



A dotted arrow ( ←-- ) points from an access category to an object it protects.

Access Categories in a Directory  
Figure 16-3



Access categories appear in a directory listing under the heading: "Access categories". The following example illustrates how access categories appear in a directory listing, displayed by the LD command:

OK, LD

<DISK>JODY>WORK (ALL), Records= 18, Quota= 18 / 0

Files= 3.

FINANCES

MEMO

REPORT

Access categories= 2.

COVER.ACAT

MEMO.ACAT

OK,

In this example, the directory WORK contains two access categories: COVER.ACAT and MEMO.ACAT. This directory listing gives no information on whether these access categories are currently protecting any object(s). (You obtain this information with the -CATEGORY\_PROTECTED option to the LD command, discussed in PRIMOS Commands Reference Guide.) Quotas are explained in Chapter 17.

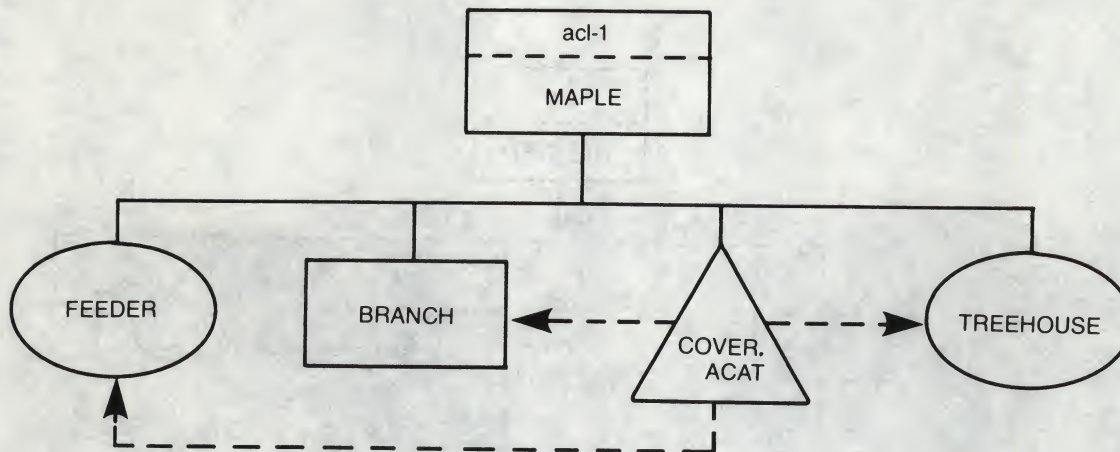
If you have a group of objects (such as a group of programs) that you want accessible to a special group of users, you can create an access category (a named ACL) and link all the objects to the access category. The use of access categories provides a convenient alternative to setting the same specific ACL individually on each of the objects; the convenience is apparent if the list of users is long or needs frequent adjustment. By changing the access rights once, in the access category, you can change the protection on all the objects that the access category protects.

If you delete the objects linked to an access category, the access category itself continues to exist as a file system object.

#### Note

It is essential that you do not confuse a specific ACL, which exists without a name of its own as an attribute of a single object, with an access category, which has a name, exists independently, and may be linked to several objects. Figure 16-4 illustrates the difference once more.





acl-1 is a specific ACL and an attribute of MAPLE; COVER.ACAC is an access category and has been linked to three objects: FEEDER, BRANCH, and TREEHOUSE.

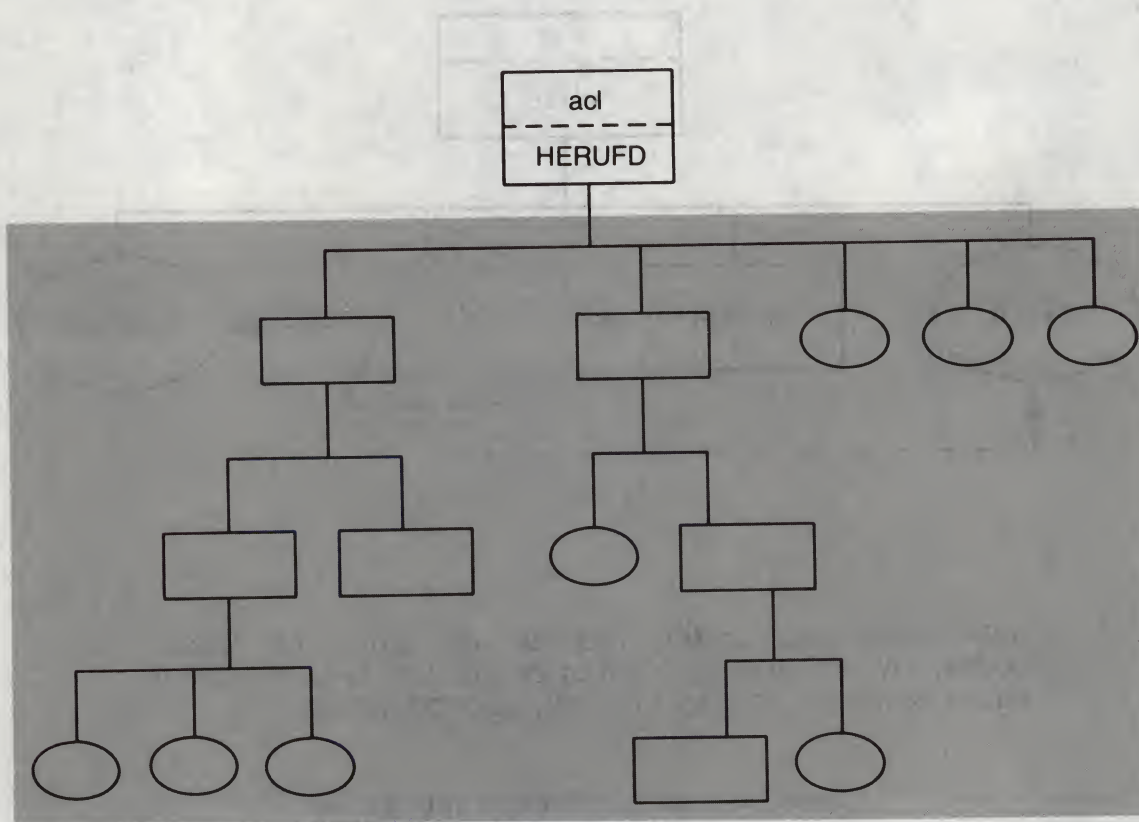
Specific ACLs and Access Categories  
Figure 16-4

### DEFAULT PROTECTION

It is not necessary to set a specific ACL or an access category on each file system object individually in order to protect it. Instead, you may use default protection that provides protection automatically. If you set a specific ACL or an access category on a directory, this ACL automatically provides default protection for all objects lower in the directory tree, unless you explicitly specify otherwise. This default feature is useful if all your work should have the same protection. In this case, an ACL need be set only once, on the top-most directory you use.



Figure 16-5 illustrates a directory tree protected by default protection. The specific ACL protecting HERUFD has automatically become the default protection for each of the other objects in the tree.



The shaded area includes all objects protected by default from the specific ACL protecting HERUFD.

A Directory Tree Under Default Protection  
Figure 16-5

#### Note

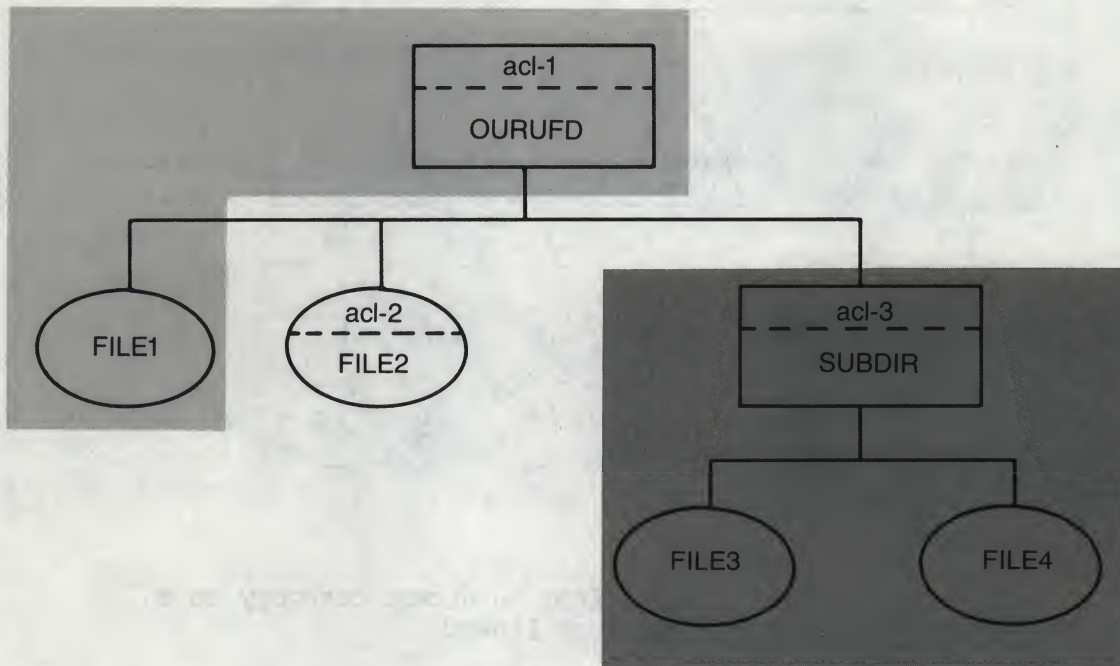
A specific ACL set on a directory does not automatically provide default protection for subdirectories that are already protected through the password protection system (discussed in Appendix F). These password directories must be explicitly converted to ACL directories. (Conversion is discussed in Appendix F.) If you create new subdirectories under the ACL directory, they will be automatically protected by default.



Providing Default Protection From Specific ACLs

Assuming you have protect (P) access, you may override the default protection by setting a specific ACL on any given object. This might be useful if you have an object (for example, a sensitive report) that you wish no one else to access, yet you want people to be able to read your other files. You can set a specific ACL on this single object that allows only you to access it. At the same time you can allow \$REST (everybody else) read (R) access to your other files through default protection.

Figure 16-6 illustrates the combined use of specific ACLs and default ACLs. OURUFD is protected by the specific ACL acl-1. FILE1 is also protected by acl-1, serving as a default ACL. FILE2 is protected by specific ACL acl-2. SUBDIR is protected by specific ACL acl-3. FILE3 and FILE4 are also protected by acl-3, providing them default protection.



Shaded areas show spheres of protection  
provided by acl-1 and acl-3.

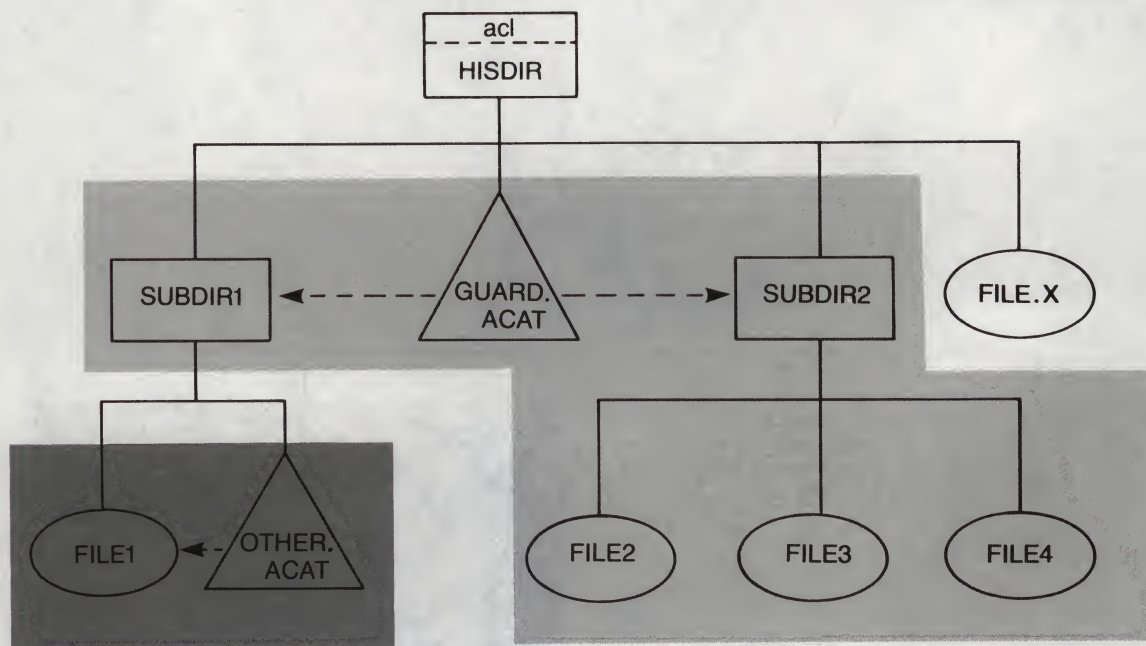
Default Protection and Specific ACLs  
Figure 16-6



### Providing Default Protection From Access Categories

Similar to specific ACLs, access categories can also provide default protection. If you set an access category on a directory, this access category automatically becomes the default protection for everything created lower in this branch of the tree, unless you explicitly specify otherwise.

Figure 16-7 illustrates the spheres of protection for two access categories. SUBDIR1 and SUBDIR2 are protected by GUARD.ACAT. FILE2, FILE3, and FILE4 are also protected by GUARD.ACAT through default protection, since it has been linked to SUBDIR2. FILE1 is protected by OTHER.ACAT, since it has been linked to SUBDIR2. FILE1 is protected by OTHER.ACAT.



Dotted arrows (←--) point from an access category to an object with which it has been linked.

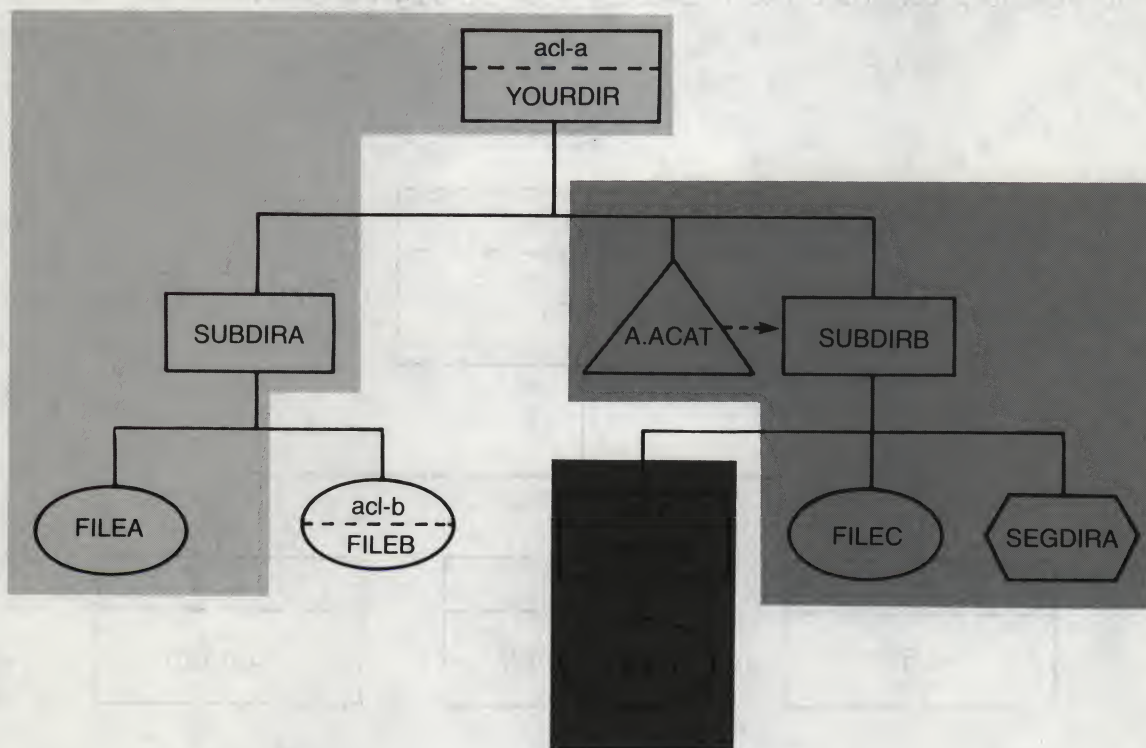
Shaded areas show the spheres of protection of the access categories.

Default Protection and Access Categories  
Figure 16-7



# Combining Default Protection, Specific ACLs, and Access Categories

All three protection mechanisms—specific ACLs, access categories, and default protection—may exist in the same directory tree, as illustrated by Figure 16-8.



Dotted arrow (--->) points from the access category to the object with which it has been linked.

Shaded areas show the spheres of protection for the specific ACLs and access category.

Default Protection, Specific ACLs,  
and Access Categories

Figure 16-8

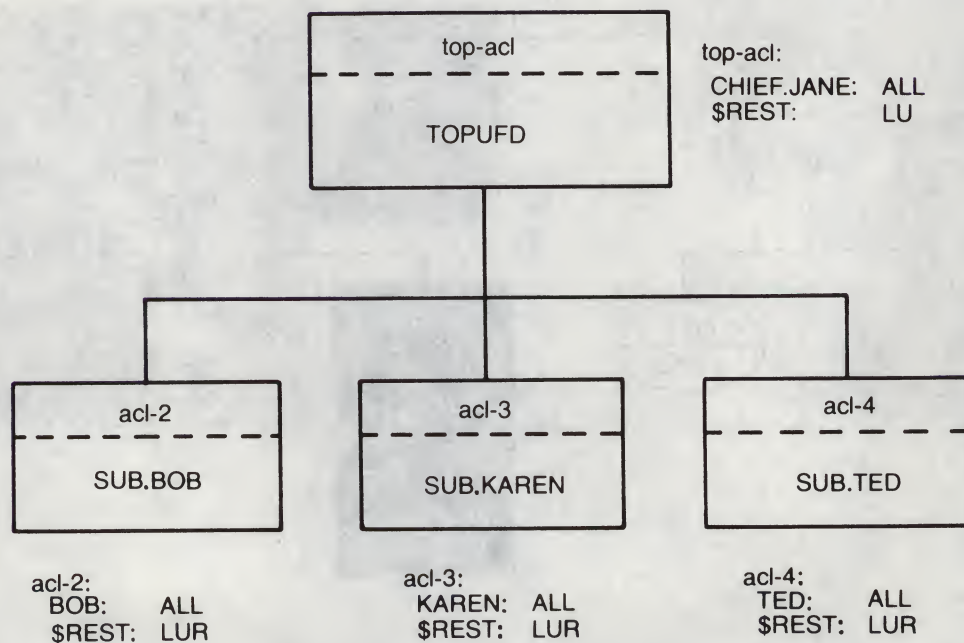


In Figure 16-8, specific acl-a protects YOURDIR explicitly and, by default, SUBDIR and FILEA. Specific acl-b protects FILEB explicitly. Access category A.ACAT protects SUBDIRB explicitly and, by default, SEGDIRA and FILEC. Specific acl-c protects SUBDIRC explicitly and, by default, FILED.

#### WHO MAY DISTRIBUTE RIGHTS

Access rights to top-level UFDs are first assigned to "owners" of these UFDs by the System Administrator (or occasionally by a Project Administrator). Since "owners" (users with protect access) can change the access of their UFDs, they can share their rights at will or designate partial rights to others anywhere in their directory tree.

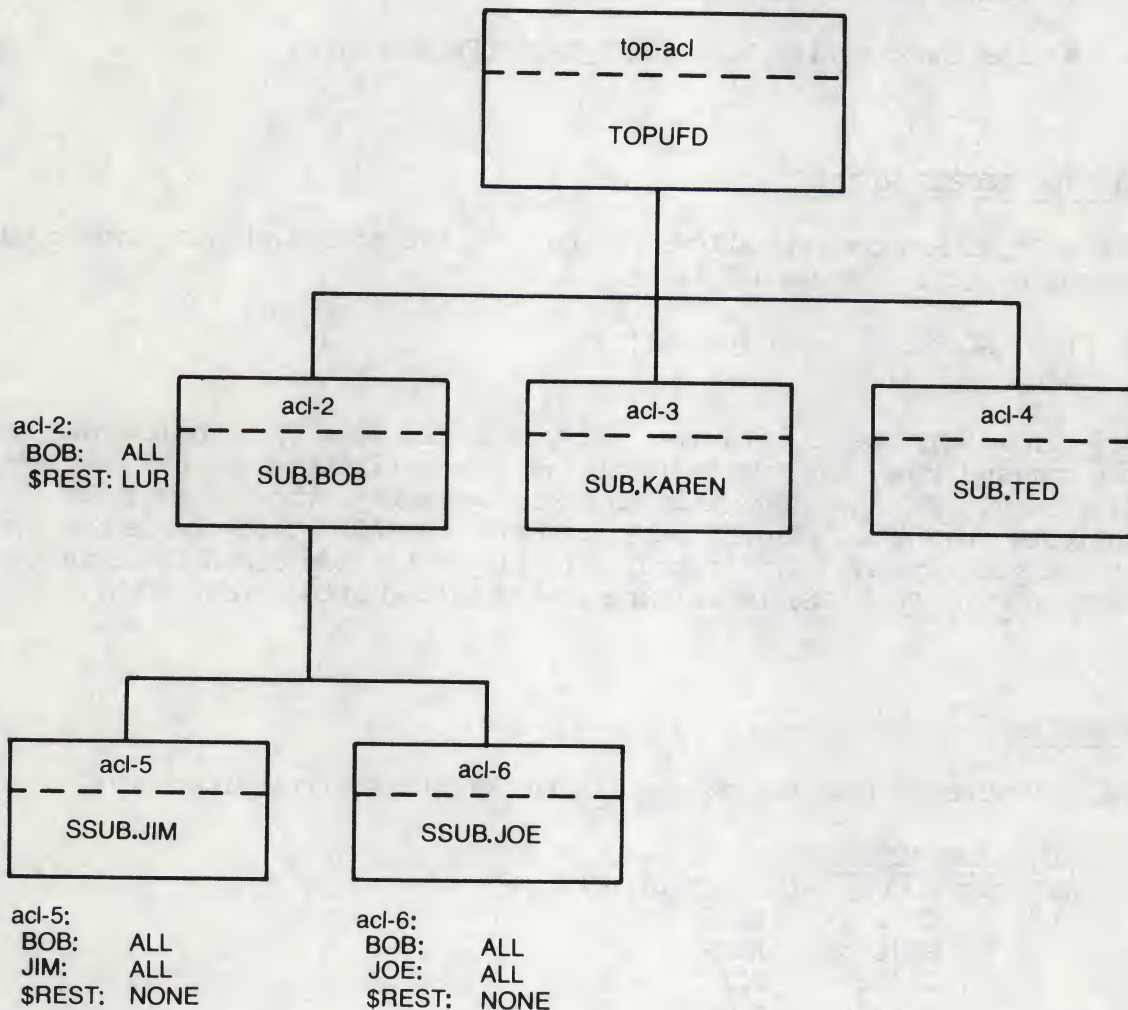
For example, consider the directory tree in Figure 16-9.



ACLs in a Directory Tree  
Figure 16-9



In Figure 16-9, CHIEF.JANE has all rights to TOPUFD. CHIEF.JANE has granted all rights in SUB.BOB, SUB.KAREN, and SUB.TED to BOB, KAREN, and TED, respectively. CHIEF.JANE herself (as part of the \$REST group) now has only list, use, and read access to SUB.BOB, SUB.KAREN, and SUB.TED. However, she could, if she wished, alter the specific ACLs on each of the subdirectories to grant herself full rights. The control available to the "owners" of the subdirectories is slightly different. BOB has full rights to his own directory (SUB.BOB) and can change rights to it at will. BOB has only list, use, and read rights to SUB.KAREN and SUB.TED, and may not change these rights. He may grant rights to subdirectories lower in his own branch of the tree, as illustrated by Figure 16-10.



More ACLs in a Directory Tree  
Figure 16-10



In Figure 16-10, BOB has granted full rights to JOE and to himself in SSUB.JOE and full rights to JIM and himself in SSUB.JIM. Everyone else (\$REST) has no rights.

### COMMANDS FOR USING ACLS AND ACCESS CATEGORIES

The next sections of this chapter will explain the commands for using ACLs and access categories. These commands explain how to:

- List access rights (LIST\_ACCESS)
- Control access to files and directories (SET\_ACCESS)
- Change access (EDIT\_ACCESS)
- Examine priority ACLs (LIST\_PRIORITY\_ACCESS)

### LISTING ACCESS RIGHTS

The LIST\_ACCESS command allows you to list the access rights connected to any object. The format is:

```
{ LIST_ACCESS }   [objectname]
{ LAC
```

objectname may be a pathname. If you do not specify an objectname on the command line, the system lists the access rights for the current directory. If the object is an access category, the system lists its contents (the ACL). In all other cases, the ACL that protects the object you specify is listed. Priority ACLs (explained later in the section PRIORITY ACLS, below) are always listed after other ACLs.

### Examples

1. You wish to list the access rights for the current directory:

```
OK, LIST_ACCESS
ACL protecting "<Current directory>":
    CAT:      NONE
    SQUIRREL: ALUR
    .BIRDS:   ALL
    $REST:    LUR

OK,
```



2. You wish to list the access rights for a subdirectory:

OK, LIST\_ACCESS BEECH>BRANCH5>TWIG42

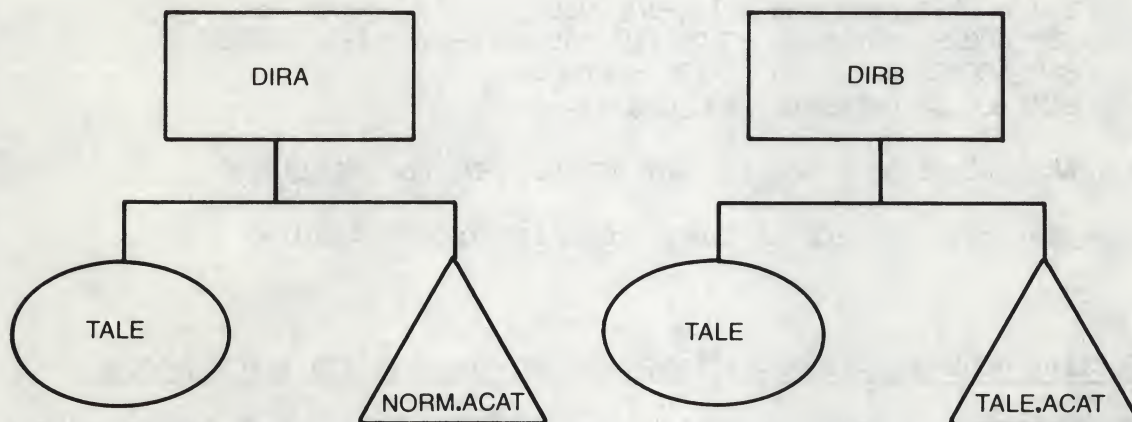
"BEECH>BRANCH5>TWIG42" protected by default ACL (from "<FOREST>BEECH"):

ROBIN: DALUR  
 .BOYS: NONE  
 .LEAVES: ALL  
 \$REST: LUR

OK,

### Specifying the .ACAT Suffix

You need not supply the ".ACAT" suffix when listing an access category, unless there is an object in the directory with the same base name as that of the category. For example, consider the two directories in Figure 16-11:



Listing Access Categories  
 Figure 16-11

To list the contents of NORM.ACAT in DIRA, you may say either of the following:

LIST\_ACCESS NORM  
 LIST\_ACCESS NORM.ACAT

To list the contents of TALE.ACAT in DIRB, you must specify:

LIST\_ACCESS TALE.ACAT

(If you say LIST\_ACCESS TALE, you will get the access rights for the file TALE.)



CONTROLLING ACCESS TO FILES AND DIRECTORIES

The SET\_ACCESS command specifies the access rights to be associated with a file or directory and controls the creation of access categories. The command has four forms.

To use the SET\_ACCESS command, you must have protect (P) access under at least one of the following conditions, as appropriate:

- To the parent directory of any object specified
- To the directory specified
- In the existing access category specified

In all cases you must have list (L) and use (U) access to the parent directory.

The syntax of the command has the following forms:

```
SET_ACCESS pathname acl [-NO_QUERY]
SET_ACCESS pathname -CATEGORY category-name [-NO_QUERY]
SET_ACCESS pathname -LIKE reference
SET_ACCESS pathname [-NO_QUERY]
```

Abbreviations: SAC for SET\_ACCESS, -NQ for -NO\_QUERY

The operation of each of these forms is discussed below.

Setting ACLs on Existing Files, Segment Directories, and Directories

To protect an existing file, segment directory, or directory with a specific access control list, give the command:

```
{SET_ACCESS} pathname acl [-NO_QUERY]
{SAC} [-NQ]
```

pathname is the name of the file, segment directory, or directory to be protected. If pathname does not exist, SET\_ACCESS will create it as an access category, as explained under Creating an Access Category below.

acl is the access control list (ACL) specifying access for the pathname. The ACL is composed of pairs of identifiers and rights. Each pair is connected by a colon in the form: identifier:rights. Do not put spaces before or after the colon. To grant multiple rights, type the letter symbols for the rights together with no intervening spaces (as "PDALU"). You may type the letter symbols in any order (such as RLU, WURALD), but they will always appear on the terminal in the order: PDALURW.



The ACL may contain up to 32 pairs, but may not in total be longer than 160 characters, including blanks. Multiple pairs are separated by spaces. The \$REST grouping, unless specified on the command line, is automatically given no rights (the designation "NONE").

If a specific ACL (as opposed to the default ACL) already exists for this object, the user will be queried before it is replaced with the new specific ACL. The query may be suppressed with the `-NO_QUERY` option.

For example, consider the following session:

```
OK, SET_ACCESS BLUEJAY JOHNNY:ALUR .DOGS:LUR
A specific ACL for "BLUEJAY" already exists.
Do you want to replace it? YES
OK,
```

The command and the "YES" answer now give JOHNNY add, list, use, and read rights to BLUEJAY. The users in the .DOGS group get list, use and read rights. Everybody else (\$REST) automatically gets no rights. If BLUEJAY had not already been protected by a specific ACL, the query would not have appeared. The new specific ACL on BLUEJAY looks like this:

```
JOHNNY:    ALUR
.DOGS:     LUR
$REST:     NONE
```

### Using Access Categories

Creating an Access Category: To create an access category, use the `SET_ACCESS` command with the following format:

```
{ SET_ACCESS } category-name acl [ -NO_QUERY ]
{ SAC          }                  [ -NO  ]
```

category-name is the name to be given to the new access category. The category-name may be given as a pathname. If this name does not end in the suffix ".ACAT", (for access category) the system will automatically append this suffix. acl specifies the pairs of identifiers and rights for the access category. Unless the `-NO_QUERY` option is given, the user will be queried to double-check his or her intent.

For example:

```
OK, SET_ACCESS PROTECT ME:ALL .GROUP:LUR
"PROTECT.ACAT" does not exist. Create access category? YES
OK,
```



The command LIST\_ACCESS PROTECT produces the following display:

Access category "PROTECT.ACAT":

```
ME:      ALL
.GROUP:  LUR
$REST:   NONE
```

#### Note

If the category-name you select for your new access category already exists in your directory as the name of a file, segment directory, or directory, then SET\_ACCESS assumes you are trying to set a specific ACL on that object, as explained earlier.

Replacing the Contents of an Access Category: If the access category already exists, the SET\_ACCESS command will replace the category's existing access list with the new access list specified on the command line. The format is identical to that for creating new access categories:

```
{ SET_ACCESS } category-name acl [ -NO_QUERY ]
{ SAC          }                  [ -NQ      ]
```

You need not include the ".ACAT" suffix when you specify the access category name.

For example, assume that you wish to change the contents of the access category PROTECT.ACAT created in the last example:

```
OK, SET_ACCESS PROTECT ME:ALL $REST:LUR
"PROTECT.ACAT" is an existing access category.
Do you want to replace it? YES
OK,
```

PROTECT.ACAT now contains the following ACL:

```
ME:      ALL
$REST:   LUR
```

Protecting Objects With Access Categories: To protect an existing file system object with an existing access category, use the SET\_ACCESS command with the following format:

```
SET_ACCESS objectname -CATEGORY category-name
```

objectname may be given as a pathname. The object and the access category must be in the same directory.

For example, using the access category PROTECT.ACAT above:

```
SET_ACCESS MYFILE -CATEGORY PROTECT
```



Deleting an Access Category: To delete an access category, whether empty or not, use the DELETE command, as for any other file system object:

DELETE category-name

If a category currently protecting an object is deleted, the access for that object reverts to the default protection.

### Setting Access Rights to Match Those on Other Objects

You can use the -LIKE option to make the access rights of one object match the rights on another object. The format is:

SET\_ACCESS objectname -LIKE reference

Both objectname and reference must be the names of existing file system objects and may be given as pathnames. If objectname is the name of an access category, the ACL it contains becomes identical to the ACL associated with the reference. If objectname is a file, directory, or segment directory, a specific ACL is set on objectname identical to the ACL associated with the reference.

For example, the file OUTLINE gives all access rights to MARY, LUR rights to .GROUP, and no rights to anyone else (\$REST). To make rights on a second file, REPORT, identical to rights on OUTLINE, give the command:

SET\_ACCESS REPORT -LIKE OUTLINE

The following access rights now exist on REPORT:

MARY:	ALL
.GROUP:	LUR
\$REST:	NONE

### Reverting to Default Access

You may have an object protected by a specific ACL or by an access category and find that you wish to change its protection back to default access. This is done by the SET\_ACCESS command in the following format:

SET\_ACCESS objectname

objectname must be a file, segment directory, or directory. It may not be an MFD. Objectname may be expressed as a pathname.



For example, consider the following sequence:

OK, LIST\_ACCESS MEMO

ACL protecting "MEMO":

\$REST: DALURW

OK, SET\_ACCESS MEMO

OK, LIST\_ACCESS MEMO

"MEMO" protected by default ACL (from "<DISK>UFDNAME"):

JEAN: ALL

\$REST: NONE

OK,

The original rights on the file MEMO (\$REST:DALURW) have been removed and the file has reverted to default access (JEAN:ALL and \$REST:NONE).

#### CHANGING ACCESS RIGHTS

The EDIT\_ACCESS command allows you to modify existing ACLs and access categories. Its format is:

```
{ EDIT_ACCESS } objectname acl [-NO_QUERY]
EDAC                [-NQ]
```

objectname may be any file system object and may be expressed as a pathname. The old access list will be modified to reflect the new rights given in each identifier:rights pair specified in acl. If you use EDIT\_ACCESS with objects that are default-protected or category-protected, you will be queried to determine whether or not you wish to create a new specific ACL. Using the -NO\_QUERY option will suppress this query.

If the identifier already exists in the ACL, its access is changed. A null access (e.g., "JOHN: ") indicates that the identifier should be removed from the list. To edit an ACL, you must have protect access on its parent directory, or protect access in the ACL itself. This means that a user may change the ACL protecting his top-level directory himself (assuming that he has protect access on it).



For example, consider the following sequence:

OK, LIST\_ACCESS REPORTS

"Reports" protected by default ACL (from "<DISK>UFDNAME"):

JACK: LUR  
STEVE: ALL  
\$REST: NONE

OK, EDIT\_ACCESS REPORTS JACK:DALURW JILL:LUR

"REPORTS" is default-protected. Create specific ACL? YES

OK, LIST\_ACCESS REPORTS

ACL protecting "REPORTS":

JACK: DALURW  
JILL: LUR  
STEVE: ALL  
\$REST: NONE

JACK's original rights (LUR) to REPORTS have been changed to "DALURW". JILL now has LUR access. The original rights of STEVE (ALL) and \$REST (NONE) remain unchanged.

#### Note

If you change the access rights on a directory to which you are attached and wish to check the change, you must reattach to the directory before the changes will take effect for you. If you change access rights on your origin directory and subsequently use the ORIGIN command, the rights will not be changed. For the ORIGIN command to reflect the new rights, you must log out and then log in again.

#### EDIT\_ACCESS vs. SET\_ACCESS

Both the SET\_ACCESS and EDIT\_ACCESS commands can be used to change existing access rights, but the two commands behave differently.

SET\_ACCESS replaces the entire existing access list with the new list given on the command line.

EDIT\_ACCESS modifies the existing access list to merge the new list given on the command line with the old list. An identifier that appears in both new and old lists is given its new rights only.

Which Command to Use: If you are making only minor changes to a long access list, EDIT\_ACCESS is probably easier to use. If you are making substantial changes (especially many deletions and few additions or modifications) SET\_ACCESS is probably easier to use.



**WARNING**

To change the access of top level UFDs, you should use EDIT\_ACCESS, not SET\_ACCESS. (If you use SET\_ACCESS and fail to include yourself in the access list, you may no longer have any rights at all to your own directory. You can create the same problem with EDIT\_ACCESS, but to do so, you must explicitly remove yourself from the ACL. If this should inadvertently happen, see your administrator.)

PRIORITY ACLS

There are times (for example, during system backups) when the System Administrator may need to control all access of the system. For this reason, the Operator or System Administrator may override any user-defined ACL by creating a priority ACL. The priority ACL defines access for the entire disk. When a priority ACL is active on a disk, its contents are always displayed by the LIST\_ACCESS command following the normal ACL.

For example:

OK, LIST\_ACCESS

ACL protecting "<Current directory>":

JOHN: ALL

.GROUP: ALL

\$REST: LUR

Priority ACL in effect for "<Current directory>":

.ADMINISTRATORS: ALL

OK,

The LIST\_PRIORITY\_ACCESS Command

Since it is possible to prevent users from accessing even the MFD with a priority ACL, the LIST\_PRIORITY\_ACCESS command allows users to read the contents of the priority ACL on any disk partition. Its format is:

```
{ LIST_PRIORITY_ACCESS } diskname
  LPAC
```



For example:

```
OK, LIST_PRIORITY_ACCESS FOREST
Priority ACL on partition "<FOREST>":
  DEER:    ALL
  $REST:   NONE
OK,
```

If no priority ACL exists on the partition, you will receive the message:

```
Priority ACL not found. <FOREST> (list_priority_access)
ER!
```

### TIPS ON SETTING ACCESS RIGHTS

Using access rights allows great flexibility of control. The following discussion will help you to select the most useful combination of access rights for various purposes.

#### More Information on Access Rights

Protect (P): Protect is the most powerful of all access rights. If a user has protect access on a directory, he may change the ACL protecting it to allow him any other rights. Since list access is required to read the directory in the first place, and use access to attach to it, protect should never be given without list and use.

Delete (D): In the ACL system, delete is a right associated with individual directories, not with individual files. Because of this, you may mark important files as "delete-protected" with the SET\_DELETE command (explained in Chapter 3). If delete access is available on a directory, any file system object immediately contained in that directory may be deleted. Delete and add accesses (explained later) together are required to rename files.

Add (A): Objects can only be created in a directory on which the user has add access. Since newly created files are opened with all file accesses available, granting add access without write access (explained below) allows users to create new files (for example, to provide a copy of a file) but not to change them after they exist. Delete and add accesses are required to rename files.

List (L): List access allows the user to list the contents of a directory.



Use (U): Use access allows a user to attach to a directory and to use the directory name in a pathname. A user may thus pass through a directory but cannot (without list access) obtain any information about the directory. Use access is available when you want to give access to an object, but also want to keep all other directory information invisible. If an object cannot be accessed for any reason, the message "No information" is returned. Use access is required on all directories, including MFDs, in order to be able to attach to them, and consequently to search them for entries. Use access, on its own, is the most restricted of accesses. However, since it controls the user's ability to attach to a directory, it should usually be granted along with other access rights to enable the other rights to work.

Read (R): Read access allows a user to read files.

Write (W): Write access allows the user to write and to truncate files.

ALL: The ALL designation grants all rights to a user. Specifying "PDALURW" instead of "ALL" on the command line will grant the same rights as "ALL". The list of letters, instead of "ALL", will appear in a listing of the ACL.

NONE: The NONE designation denies all access. The \$REST group is automatically given NONE as access rights to protect your work from intrusion.

### Open Systems and Closed Systems

If you want a totally open system, where all users have all rights to all work, the System Administrator can set the following ACL on each top UFD:

```
$REST:  ALL
```

Conversely, if you want a tightly controlled system where only the owner of a UFD has full rights and no one else can do anything, the System Administrator can set the following ACL on each top UFD:

```
owner:  ALL
$REST:  NONE
```

owner is a user id.

Most systems will want to establish protection schemes somewhere between these two extremes. Factors you may wish to consider are presented in the rest of this chapter.



### Rights for the "Owner" of a Directory

Normally "owners" of directories will be granted all rights (that is, "PDALURW", or "ALL"). If the "owner" of the directory is really only supposed to use it, however, sometimes protect permission will be omitted, resulting in DALURW rights. This allows the user of the directory to do anything but change protection and attributes of objects in the directory and its subtree.

### Rights for Other Users in a Group or Project

When several people work together on a project, it is often useful to allow them to access each other's files, but not to change or destroy them. In such cases the combination LUR is helpful. Users with LUR rights may examine files and directories and may execute programs, but they may not change anything.

### Rights for Outsiders

True outsiders are often denied rights altogether with the NONE designation. Sometimes, however, you may wish certain files to be accessible to anyone. In this case, granting UR access allows user to read files whose names are known in advance. UR rights do not allow users to modify anything, and do not allow the directory to be perused for additional entries.

### Passing Information Around

Sometimes it is useful to allow users to "pass" files back and forth, but at the same time the "owner" does not want his other files to be damaged. In this situation, the user can grant ALUR rights; ALUR rights allow new files to be created and written to (since new files are always opened with RW rights), but do not allow existing files to be modified.

### ACL SUBROUTINES

Several subroutines that manipulate ACLs are available. These subroutines allow you to access the ACL system from programs and to set, modify, list, and remove protection on objects just as you would from the terminal. For details on these subroutines, see the Subroutines Reference Guide.







# 17

## Customizing Your Environment

### INTRODUCTION

Users can customize their command environment in several ways:

- They can use the RDY command to choose the form of prompts to be displayed at their terminal during an interactive session or in a command file.
- They can use the ABBREV command to define their own abbreviations for PRIMOS commands, and to use those abbreviations during interactive sessions.
- They can define global variables that can be used at PRIMOS level and by user programs.
- They can create a special command file that will automatically, at login, establish operating features that they use regularly in interactive sessions.
- They can send messages and set their terminal's ability to receive messages with the MESSAGE command.
- They can set maximum limits, or quotas, on the amount of storage space their subdirectories are allotted on a disk.

The commands and procedures for customizing the command environment are described in this chapter.



CHANGING THE PROMPT MESSAGE (RDY)

Instead of its normal brief OK, and ER! prompts, PRIMOS can also supply a long form of prompt message. The display format of the long prompt is:

```

{ OK   }
{ ER   }   time   CPU-time   I/O   [command level]
{ text }
```

The message displays OK (for ready), ER (for error), or a user-defined text, followed by the time (hh:mm:ss, 24-hour clock), the amount (in seconds) of CPU time and I/O time used since the last prompt, and sometimes the user's command level. (The command level is displayed only if it is greater than 1; most users don't need to worry about it.) For example:

```

OK 16:23:25  0.024  0.021
ER 10:07:31  0.100  0.609
```

Users can change the form of prompt message displayed at their terminal by giving the RDY command.

You may wish to change your prompt message if you work on several systems within a network and want a convenient way to keep track of which system you are currently using. You can assign a different prompt to each system.

The format for the RDY command is:

```
RDY [options]
```

If given without options, the RDY command prints a single long-form prompt. If given with options, the command changes the form and/or content of the prompt as summarized in Table 17-1.

The new prompt message, text, may be up to 20 characters long. If it contains special characters or embedded blanks, it must be enclosed in single quotation marks.

More than one option can be given with the same RDY command.

The following example illustrates the use of RDY options:

```

OK, RDY -LONG
OK 13:11:41  0.827  1.739
RDY -OFF
RDY -ON
OK 13:11:56  0.066  0.000
RDY -RL Absolutely!
Absolutely! 13:12:01  0.054  0.000
RDY -BRIEF -RB GO!
GO!
```



Table 17-1  
Options for the RDY Command

Option	Function
-LONG	Sets the terminal to the long form of prompt.
-BRIEF	Returns it to the standard "OK,".
-OFF	Suppresses prompts entirely.
-ON	Reactivates the previous version of the prompt (long or brief).
{ -READY_LONG text } { -RL text }	Changes the text portion of the long ready message to <u>text</u> . Default text at login time is "OK".
{ -READY_BRIEF text } { -RB text }	Changes the brief ready message to <u>text</u> . Default message at login time is "OK,".
{ -ERROR_LONG text } { -EL text }	Changes the text portion of the long error message to <u>text</u> . Default text at login time is "ER".
{ -ERROR_BRIEF text } { -EB text }	Changes the brief error message to <u>text</u> . Default message at login time is "ER!".

#### CREATING AND USING ABBREVIATIONS (ABBREV)

The PRIMOS command ABBREV allows you to create your own abbreviations for use in PRIMOS command lines. Its form is:

ABBREV [pathname] [options]

To use ABBREV, you:

- Create an empty abbreviation file.
- Define abbreviations within the file.
- Invoke ABBREV to activate the file during any work session in which you want to use your abbreviations.

When an abbreviation file is activated, PRIMOS calls its abbreviation processor to scan each PRIMOS command entered from the user's terminal. The abbreviation processor checks each word against the active abbreviation file, expands all abbreviations to their full form, then passes on the commands to the standard command processor. You can modify your abbreviation file at any time; but you can use it only for



interactive sessions or CPL programs. Abbreviations will not be expanded inside COMINPUT files. Once your abbreviation file is activated, it remains active until you either activate a different abbreviation file, give the ABBREV -OFF command, or log out.

### Creating an Abbreviation File

Invoke the ABBREV command with the -CREATE option, giving a pathname which names and locates the new file:

```
ABBREV pathname -CREATE
```

This command creates and activates an empty abbreviation file. Therefore, the file specified must not already exist. If the abbreviation file is to be located in the current directory, you need give only the final element of the pathname (the simple filename).

For example, assume you are attached to a UFD named UFD#1. The following command will create an abbreviations file named UFD#1.ABBREV in UFD#1:

```
ABBREV UFD#1.ABBREV -CREATE
```

### Defining Abbreviations

You may define abbreviations and put them into your active abbreviation file by using the -ADD\_COMMAND, -ADD\_ARGUMENT, and -ADD options of the ABBREV command. The format is:

```
ABBREV [pathname] { -ADD_COMMAND  
                   -ADD_ARGUMENT } name value  
                   -ADD
```

Abbreviations: -AC for -ADD\_COMMAND; -AA for ADD\_ARGUMENT

name is the abbreviation. value is the full reference (including arguments and options, if any) that will be substituted for the abbreviation. If pathname is omitted, the abbreviation is added to the abbreviation file currently active for your user id.

The -ADD\_COMMAND Option: The -ADD\_COMMAND option adds an abbreviation that will be expanded to its full reference only when it occurs in the command position of a command line. For example:

```
ABBREV -ADD_COMMAND JD JOB -DISPLAY
```



This example enters the abbreviation "JD" in the user's abbreviation file, and defines it as standing for the command "JOB" plus the option "-DISPLAY." Whenever this abbreviation file is activated during a work session at a terminal, typing "JD" at that terminal will be equivalent to typing "JOB -DISPLAY".

The -ADD\_ARGUMENT Option: The -ADD\_ARGUMENT option adds an abbreviation that will be expanded to its full reference only when it occurs in the argument position of a command line. For example:

```
ABBREV -ADD_ARGUMENT F THIS_IS_A_LONG_PROGRAMNAME
```

When F occurs in the argument position on a command line, as in:

```
SEG F
```

the F will be expanded into THIS\_IS\_A\_LONG\_PROGRAMNAME. The command line above would become:

```
SEG THIS_IS_A_LONG_PROGRAMNAME
```

The -ADD Option: The -ADD option adds an abbreviation that will be expanded to its full reference when it occurs anywhere on the command line.

Although the -ADD option creates an abbreviation that will expand as either a command or as an argument, the more specific options -ADD\_COMMAND and -ADD\_ARGUMENT are usually more useful.

-ADD\_COMMAND vs. -ADD: To define command abbreviations, you should ordinarily use the -ADD\_COMMAND option, rather than the -ADD option. This will prevent file system objects (or other arguments) which happen to have the same name as the command abbreviation from being expanded, incorrectly, into the full command name.

For example, you use the -ADD option to define P as the abbreviation for SLIST:

```
ABBREV -ADD P SLIST
```

You also have a file named P in your directory. If you try to edit the contents of the file P, using:

```
ED P
```

you will get the following error message (provided you do not also have a file named SLIST):

```
Not found. SLIST (OPENR)
ER!
```



What has happened is that the command processor has substituted "SLIST" for "P". In effect your command line says "ED SLIST" and you have no file named "SLIST". You cannot access file P until you change your abbreviation P.

If you use the -ADD\_COMMAND option to define P:

```
ABBREV -ADD_COMMAND P SLIST
```

then the command:

```
ED P
```

will work correctly and put you into EDIT mode within the Editor so you can edit the contents of the file P. The command processor in this case has not expanded the "P", because it does not occur in the command position on the command line.

-ADD\_ARGUMENT vs. -ADD: To define abbreviations for file system objects (or other arguments), you should ordinarily use the -ADD\_ARGUMENT option, rather than the -ADD option. This will prevent commands that have the same name (or PRIMOS abbreviation) as your object's abbreviation from being expanded, incorrectly, into the full object name.

For example, you use the -ADD option to define A as the abbreviation for the long filename ASSOCIATES\_OF\_THE\_FIRM that you reference frequently:

```
ABBREV -ADD A ASSOCIATES_OF_THE_FIRM
```

A is also the PRIMOS abbreviation for the ATTACH command. If you try to attach to another directory using the ATTACH abbreviation, as:

```
A ANOTHER_UFD
```

you will get the following error message:

```
Not found. ASSOCIATES_OF_THE_FIRM (std$cp)
ER!
```

The command processor has given your abbreviation precedence and has assumed you are giving a command named ASSOCIATES\_OF\_THE\_FIRM, not ATTACH.

If you use the -ADD\_ARGUMENT option to define A:

```
ABBREV -ADD_ARGUMENT A ASSOCIATES_OF_THE_FIRM
```

then the A abbreviation for the ATTACH command will work correctly. The A abbreviation for the filename will be expanded only when the abbreviation occurs in an argument position on the command line.



Abbreviations for Multiple Commands: A single abbreviation may also stand for a series of commands to be executed sequentially, as a unit. You define value for such a series by separating each full command reference (together with its arguments and options) from the next by a semi-colon (;). For example:

```
ABBREV -ADD AL ATTACH SUBDIR2; LD
```

AL is now defined as an abbreviation that will attach you to the subdirectory SUBDIR2 and list the directory.

### Activating an Abbreviation File

```
ABBREV pathname [-ON]
```

activates an existing abbreviation file. PRIMOS loads the abbreviation table from the specified file and checks each word typed at the terminal against the abbreviations in the file. PRIMOS expands the abbreviations it finds into their full form before giving commands to the command processor.

### Using Variables in Abbreviations

You can define variables within an abbreviation by using numerals flanked by ABBREV's escape character, %. The symbol %1% stands for the first word following the abbreviation when it is actually issued as a command; %2% stands for the second word, and so on. (Currently, up to nine variable words are allowed.) This feature is particularly useful for commands naming files. For example, defining an abbreviation by the command:

```
ABBREV -ADD F %1% %2% -L %2%.LIST -XREF -64V -DEBUG
```

would allow the abbreviation processor to translate the command:

```
F FTN DRAGON
```

into the command:

```
FTN DRAGON -L DRAGON.LIST -XREF -64V -DEBUG
```

Similarly,

```
F F77 DRAGON
```

would become:

```
F77 DRAGON -L DRAGON.LIST -XREF -64V -DEBUG
```



Other Options: ABBREV has options for refining definitions, changing or deleting definitions, and so on. An additional use of the -ON option and three other commonly used options are:

<u>Command</u>	<u>Function</u>
<u>ABBREV</u> -OFF	Deactivates abbreviations file.
<u>ABBREV</u> [pathname] -ON	Reactivates abbreviations file. If pathname is not supplied, the abbreviations file previously active is reactivated.
<u>ABBREV</u> [pathname] -DELETE abbrev-1 [...abbrev-n]	Deletes the named abbreviations from the file.
<u>ABBREV</u> [pathname] -LIST	Lists the contents of the abbreviations file specified by <u>pathname</u> . If <u>pathname</u> is given and the file is deactivated, the -LIST option also activates the file. If <u>pathname</u> is omitted, the -LIST option lists the abbreviations file currently active. Abbreviations added with -ADD_COMMAND are preceded in the listing by "(C)", those added with -ADD_ARGUMENT by "(A)", and those added with -ADD by nothing.

For a full list of options and their uses, see the PRIMOS Commands Reference Guide.

Syntax Suppression and Abbreviations: You may wish to define an abbreviation that contains a function or global variable, but you do not wish the evaluation to be performed immediately. You can use the tilde (~) before the command line to suppress immediate processing so that the full expansion may be recorded literally. For example:

```
~ABBREV -ADD LOG LOG.[DATE -TAG]
```

This command will add the abbreviation LOG to your abbreviations file. LOG will be expanded to "LOG.[DATE -TAG]" and evaluated each time you use it. On 10 May 1982 LOG would be evaluated as LOG.820510, since the function [DATE -TAG] is evaluated as YYMMDD. (Functions are discussed in the PRIMOS Commands Reference Guide.)



USING GLOBAL VARIABLES

Sometimes you want to define variables that can be known to, and possibly modified by, a group of programs, rather than a single program. At these times, you can use global variables. Global variables are stored in one or more files inside your UFD or subdirectory. When you activate a global variable file, you can use its variables and set new ones in the following ways:

- interactively, in PRIMOS commands
- through CPL programs
- through programs written in some high-level languages

Global variables survive program termination and logouts. Once defined, they last until you delete them.

The PRIMOS commands governing global variables are shown below. For complete details see the CPL User's Guide or the PRIMOS Commands Reference Guide.

<u>Command</u>	<u>Function</u>
DEFINE_GVAR pathname -CREATE	Creates and activates an empty global variable file.
DEFINE_GVAR pathname	Activates an existing global variable file.
SET_VAR name [:=] value	Defines a new variable and places it in the active global variable file, or changes the value of an existing variable. <u>name</u> is any legal variable name, up to 32 characters long. Names of global variables must begin with a dot (.), as in ".ALPHA". <u>value</u> is an alphanumeric character string. The assignment symbol (:=) is optional.
LIST_VAR [var_name...]	Lists the variables and values contained in an active global variable file.
DELETE_VAR var_name...	Deletes (a) variable(s) from an active global variable file.



Example

The following sequence illustrates the use of these commands:

1. A user decides to create a global variable file and to name it VARFILE. She creates it with the command:

```
OK, DEFINE_GVAR VARFILE.GVAR -CREATE
```

2. She places two global variables in the file, using the SET\_VAR command:

```
OK, SET_VAR .HOME BEECH>BRANCH5>TWIG3
OK, SET_VAR .AWAY BEECH>BRANCH2>TWIG4
```

3. She checks the variables with the LIST\_VAR command:

```
OK, LIST_VAR
.AWAY          BEECH>BRANCH2>TWIG4
.HOME          BEECH>BRANCH5>TWIG3
```

4. She uses the variables in the command:

```
OK, FIN %.HOME%>FOO.FIN -B %.AWAY%>FOO.BIN
```

5. She deletes one of the global variables:

```
OK, DELETE_VAR .AWAY
OK,
```

Note

At command level, you define global variables with the SET\_VAR command. CPL programs define them with the &SET\_VAR directive or with the SET\_VAR command (but not with the &ARGS directive). High level programs define global variables through the GV\$SET routine, and reference them through the GV\$GET routine. The CPL User's Guide contains details.

CREATING LOGIN FILES

You may wish to create a special CPL program file in your origin directory that will execute each time you log in. Such a file automatically establishes features that you use regularly. This file should be called:

```
LOGIN.CPL
```



Use the Editor to create this file and enter the commands, one per line, that will provide the features you want. Some commands that you may wish to include are:

<u>Command</u>	<u>Result</u>
RDY -READY_BRIEF text	Changes the brief ready prompt to <u>text</u> . Default prompt at login is "OK,". The abbreviation for -READY_BRIEF is -RB.
RDY -ERROR_BRIEF text	Changes the brief error prompt to <u>text</u> . The default prompt at login is "ER!". The abbreviation for -ERROR_BRIEF is -EB.
ABBREV pathname	Activates your abbreviations file. <u>pathname</u> is the name of the abbreviations file. If this file resides in the login directory, only the final element of <u>pathname</u> (the simple filename) need be specified.
TERM -XOFF	Enables the CONTROL-S key to halt terminal output temporarily and the CONTROL-Q key to resume terminal output from the point of suspension.
TERM -ERASE character	Replaces the system erase character " with a character of your choice. (Backspace is useful as an erase character.)
TERM -KILL character	Replaces the system kill character ? with a character of your choice.
SLIST pathname	Prints the contents of a file at the terminal. <u>pathname</u> can be used as a message file.
DEFINE_GVAR pathname	Activates a global variable file.

Multiple options for the same command may be specified on one line. For example:

```
RDY -RB GO> -EB NO!
```

More detailed information on the RDY, ABBREV, and DEFINE\_GVAR commands appears earlier in this section. Information on the SLIST command appears in Chapter 3 and on the TERM command in Chapter 2.

Your login file may also contain any other commands that are useful to you.



When you log in, PRIMOS will search your origin directory for a file named LOGIN.CPL. If one exists, PRIMOS will execute the commands in it and then give you the ready prompt -- either the system "OK," or a prompt you have defined for yourself. If you want PRIMOS to print the contents of the file on the screen as it executes the file, include the following line as the first line in the file:

```
&DEBUG &ECHO COM
```

This is a Command Procedure Language (CPL) directive. More information on CPL appears in Chapter 9 and in the CPL User's Guide.

### Example

Suppose that your user id is JANE and that you are attached to your origin directory, the UFD BEECH. The abbreviations file FILE.ABBREV resides in BEECH. The following session will create a login file in BEECH:

```
OK, ED
INPUT
&DEBUG &ECHO COM
RDY -READY_BRIEF GO> -ERROR_BRIEF NO!
ABBREV FILE.ABBREV
TERM -XOFF
TERM -KILL #
(CR)
EDIT
FILE LOGIN.CPL
OK,
```

When you next log in, you would receive the following display on the screen: (Neither a login password nor project-id is required in this example.)

```
OK, LOGIN JANE
```

```
JANE (user 65) logged in Thursday, 24 Jun 81 10:16:36.
Welcome to PRIMOS version 19.0
Last login Wednesday, 24 Jun 81 15:15:21.
```

```
RDY -READY_BRIEF GO> -ERROR_BRIEF NO!
ABBREV FILE.ABBREV
TERM -XOFF
TERM -KILL #
GO>
```

In this example, the &DEBUG &ECHO COM command, which does not itself print at the terminal, instructs PRIMOS to print at the terminal subsequent commands. The RDY -READY\_BRIEF GO> command changes the "OK," prompt to "GO>". The ABBREV FILE.ABBREV command activates the existing abbreviations file FILE.ABBREV. The TERM -XOFF command allows



terminal output to be temporarily halted by typing CONTROL-S and resumed by typing CONTROL-Q. The TERM -KILL # command changes the kill character to the number character. Note that the ready prompts are not listed as the CPL program runs.

### Alternate Names and Forms for the Login File

Instead of a CPL program, you can create a login file called one of the following:

```
LOGIN.COMI
LOGIN.SAVE
```

LOGIN.COMI: LOGIN.COMI is a COMINPUT command file formed with the Editor in the same way as is LOGIN.CPL. LOGIN.COMI admits nearly the same commands and executes in nearly the same way. The contents of LOGIN.COMI print automatically unless PRIMOS is instructed otherwise, as explained below. LOGIN.COMI does not use &DEBUG &ECHO COM. It may use the following special commands:

```
COMOUTPUT -NTTY
COMOUTPUT -TTY
COMINPUT -END
COMINPUT -TTY
```

These commands perform the following functions:

COMOUTPUT -NTTY	Turns off screen output, including the ready prompt. Without this command, the LOGIN.COMI file displays at the terminal when you log in. If you do not want the login file to print, place this command first in the login file. Use this command together with COMOUTPUT -TTY.
COMOUTPUT -TTY	Turns on screen output. This is the default state. If you have included COMOUTPUT -NTTY at the beginning of your login file, you should also include COMOUTPUT -TTY at the end of the file, just before the COMINPUT -END or COMINPUT -TTY command.
COMINPUT { -END - TTY }	Either option instructs PRIMOS to begin executing commands entered at the terminal. This command should be the final line of the file. If it is omitted, PRIMOS will still return to the terminal to receive subsequent commands but will display the message: "End of file. Cominput. (Input from terminal.)".



The following example of a LOGIN.COMI file illustrates the positioning of these options:

```
GO> SLIST LOGIN.COMI
COMOUTPUT -NTTY
RDY -RB GO> -EB NO!
ABBREV FILE.ABBREV
TERM -XOFF
TERM -KILL #
COMOUTPUT -TTY
COMINPUT -END
GO>
```

LOGIN.SAVE: To use LOGIN.SAVE, you must write a program in FIN or some other language that produces R-mode code. The program contains whatever features you want at login. You must then compile the program and load it with the LOAD loader. This will produce an R-mode, executable binary code file, which should be called LOGIN.SAVE. LOGIN.CPL and LOGIN.COMI are easier to maintain as login files than LOGIN.SAVE because they do not have to be recompiled and reloaded when they are changed.

### SENDING MESSAGES

The MESSAGE command is used to send or receive one-line messages. Either users or the operator may send messages. Messages may be sent:

- From any user terminal to any other user terminal.
- From any user terminal to the supervisor terminal.
- From the supervisor terminal to all users.
- From the supervisor terminal to a specified user.

### User Messages

The format of a user-to-user or user-to-operator message is:

```
MESSAGE { user-id } [-NOW] [-ON systemname]
        { -usernumber }
one-line text of message
```

To send a message to the operator, omit the user-id argument. To send a message to another user, give either the user's user-id (the name under which he or she logged in) or usernumber (a number assigned by PRIMOS to the user at login). To get a list of user ids and their usernumbers, issue the STATUS USERS command (explained in Appendix G), or the MESSAGE -STATUS command (explained later under Querying Receive States).



To send a message to a user logged into another system in your network, specify the user id together with the `-ON systemname` option (where systemname is the name of the other user's system).

If you send a message to a user-id, all users logged in as that name receive the message. If you send a message to a usernumber, only the single user with that number receives the message.

one-line text of message is the single-line message to be sent (maximum 80 characters). Sending a message produces two lines of information on the receiver's terminal. The top line contains information about the sender; the second contains the text of the message. The format is:

```
*** sendername (user sendernumber [on systemname]) at hh:mm
one-line text of message
```

sendername is the sender's user id. sendernumber is the number of the sender's terminal. systemname is the name of the system the sender is using. systemname will appear only if you are working on a system where two or more processors have been linked into a configured network. hh:mm is the time of day in hours and minutes that the message was sent (24 hour clock). For example:

```
*** BEECH (user 66 on SY3) at 15:16
HELLO TO MAPLE.
```

If the `-NOW` option is specified, the message is printed immediately on the receiver's terminal.

If the `-NOW` option is not specified, messages are stored in a buffer and printed when the receiver returns to PRIMOS command level.

### Setting Receive States

Users may set the receive state of their terminal with the MESSAGE command. One of three different states may be selected to control the flow of messages:

<u>Option</u>	<u>Function</u>
<u>MESSAGE -ACCEPT</u>	Enables reception of all messages. This is the default state.
<u>MESSAGE -DEFER</u>	Inhibits immediate messages. (Messages sent with <code>-NOW</code> option will be rejected by MESSAGE. Messages sent without the <code>-NOW</code> option will be received when you return to command level.)
<u>MESSAGE -REJECT</u>	Inhibits all messages.



Deferring or rejecting messages is useful when you do not want messages to interrupt a terminal session. This can be critical in situations where you are printing the contents of a file, for example. Deferring or rejecting messages in this instance would prevent the message from being printed along with your file's contents.

You cannot send a message while in MESSAGE -REJECT mode, and you cannot send an immediate message (using the -NOW option) while in MESSAGE -DEFER mode. These restrictions exist because the receiver will not be able to respond.

### Querying Receive States

You may determine how the receive state of a user's terminal has been set with the -STATUS option of the MESSAGE command. Issuing MESSAGE -STATUS lists user ids, terminal numbers, and receive states.

The formats are:

```
MESSAGE -STATUS [user-id user number] [-ON systemname]
```

```
MESSAGE -STATUS ME
```

The command lines have the following meaning:

<u>Command</u>	<u>Function</u>
MESSAGE -STATUS	Lists the receive state of all users on your system.
MESSAGE -STATUS user-id	Lists the receive state of all users with the name <u>user-id</u> on your system.
MESSAGE -STATUS usernumber	Lists the receive state of the user with the number <u>usernumber</u> on your system.
MESSAGE -STATUS ME	Lists the receive state of your own terminal.

To query the receive state of a user or users logged into another system in your network, specify the -ON systemname option with any of the first three command lines listed above. systemname is the name of the user's (or users') system.



Error Messages

The following are common error messages sent by the MESSAGE command:

- Improper command usage or arguments. (MESSAGE)

This message may mean:

1. You have made a typing error.
2. The usernumber you have specified to receive a message does not represent a logged-in user.
3. You have used an illegal or unknown option.

- \*\*\* Unknown addressee.

The user id you have specified to receive a message is not the id of a logged-in user.

- \*\*\* User usernumber not receiving now. (MSG\$)

This message may mean one of two things:

1. If you are trying to send an immediate message (M -NOW), it means that the recipient's receive state is either DEFER or REJECT.
2. If you are sending a message without the -NOW option, this notice means that the recipient's receive state is REJECT.

- \*\*\* Unknown node.

You have specified a system name unknown to your network.

- \*\*\* User usernumber busy, please wait.

The user already has a deferred message waiting. Only one deferred message is allowed. You must send your message again.

- \*\*\* Requires -ACCEPT enabled.

You must issue MESSAGE -ACCEPT before sending a message.



- \*\*\* Requires -ACCEPT or -DEFER enabled.

You must issue MESSAGE -ACCEPT or MESSAGE -DEFER before sending a message.

## USING DISK QUOTAS

### Introduction

To ensure equitable sharing of disk storage, administrators and users can set limits (called quotas) on the amount of storage space that directories can occupy on a disk. You can set a quota on a directory if you have protect access rights (for systems that use ACLs) or owner rights (for systems that use directory passwords) to the next higher directory. Normally, this means that only the system administrator can set or change quotas on top-level UFDs. Individual users can set or change quotas on their own subdirectories, if they wish to provide personal checks on their own storage use. The commands for using quotas allow users to:

- Set a maximum storage quota on a subdirectory (SET\_QUOTA)
- Change an existing quota (SET\_QUOTA)
- Examine existing quotas and current storage use (LIST\_QUOTA, LD, SIZE)

These commands are discussed in the rest of this chapter.

### Measuring and Allocating Storage Space

Storage space is measured in disk records. A record can contain up to 1024 user data words. Thus, the number of records in a file system object equals the total number of data words in the object divided by 1024 and rounded up to the next whole number. However, a zero-length object (such as an empty directory or file) always contains one record. All numbers are decimal.

If you create a new subdirectory, its quota will initially be set to zero; that is, it has no maximum quota. However, any maximum quota that may exist on a higher directory will limit the actual storage allowance on the subdirectory. If no limit exists on any higher directory, then its storage capacity is limited only by the physical capacity of the disk with which it is associated. (Note that a quota of zero does not signify that the directory is allowed no storage at all; rather it signifies the reverse).



Setting Quotas on Directories

To set maximum storage quotas on your sub-UFDs, use the SET\_QUOTA command. The format is:

```
{ SET_QUOTA } pathname -MAX number
{ SQ }
```

**pathname** The pathname of the directory having its quota set. If you want to set a quota on the current directory, you must use the full pathname. If you want to set a quota on a subdirectory within the current directory, you need specify only the simple name (the final element of the full pathname).

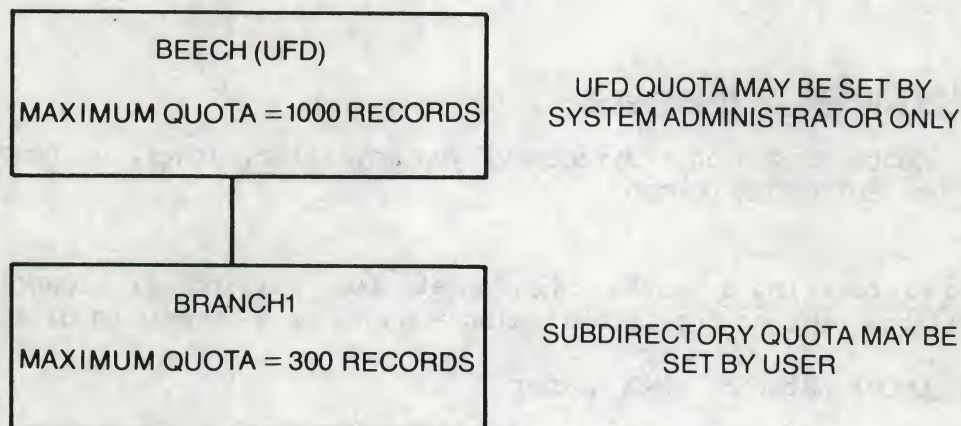
**-MAX number** The maximum number of records the directory can use.

Remember that to set a quota on a subdirectory, you must have protect rights (in ACL systems) or owner rights (in directory-password systems) to the next higher directory. That is, you must have the appropriate rights to the directory that contains the subdirectory whose quota is to be set. If you give the SET\_QUOTA command without having these rights, PRIMOS will return the message "Insufficient access rights". The quota will not be set.

For example, if you have protect rights to the UFD BEECH, you can say:

```
SET_QUOTA BEECH>BRANCH1 -MAX 300
```

Figure 17-1 illustrates the effects of this command. The System Administrator has set a quota of 1000 records on the UFD BEECH. The user has set the quota of 300 on the subdirectory BRANCH1.



Quotas Set on Directories  
Figure 17-1



Note

When you try to set a quota on a directory under a certain set of conditions, you will receive the error message:

```
File in use. directory-name (set_quota)
ER!
```

This message will appear when both of the following conditions are true:

- when the directory has no current quota (that is, quota = 0)
- when there are attached users or open files in the the directory or its subtree.

You may be able to remedy this situation. If you are the only user of the directory subtree, ATTACH to another UFD or to a level higher in the tree than the directory whose quota you wish to set. The SET\_QUOTA command should now execute.

For example, suppose you are in BEECH>BRANCH6>TWIG23. You wish to set a quota of 200 on TWIG23, which currently has no quota:

```
SQ BEECH>BRANCH6>TWIG23 -MAX 200
File in use. TWIG23 (set_quota)
ER! A BEECH
OK, SQ BEECH>BRANCH6>TWIG23 -MAX 200
OK,
```

If you are not the only active user of the directory subtree, you must wait for all other users to ATTACH elsewhere or to LOGOUT before you can set the quota.

Modifying Quotas on Directories

Once a quota exists on a directory, you may raise, lower, or remove it with a new SET\_QUOTA command.

Raising or Lowering a Quota: The format for raising or lowering a quota is the same as for establishing a quota on a non-quota directory:

```
SET_QUOTA pathname -MAX number
```



Removing a Quota from a Directory: You may remove an existing quota from a directory by setting the quota to zero. Any of the following command formats will set a directory quota to zero:

```
SET_QUOTA pathname
SET_QUOTA pathname -MAX
SET_QUOTA pathname -MAX 0
```

### Examining Quotas and Current Storage

You may wish to examine the quota on a directory and the current storage space used by directories, files, and segment directories. The LIST\_QUOTA, LD, and SIZE commands provide this information.

Using LIST\_QUOTA: The LIST\_QUOTA command tells you (1) the maximum quota on a directory; (2) the total number of records used by the entire subtree beginning with and including the designated directory; (3) the number of records used by this particular directory. The format of the command is:

```
{ LIST_QUOTA } [pathname] [-BRIEF]
  LQ
```

pathname gives the name of the directory on which quota information is requested. If pathname is omitted, the quota information on the current directory is listed. To use the LIST\_QUOTA command, you must have list (L) access to the target directory or its parent and use (U) access to all higher directories.

For example, to list the quota information on the current directory REPORTS:

OK, LQ

Maximum records allowed on "<Current directory>" = 200.

Total records used = 178.

Records used in this directory = 65.

OK,

If REPORTS were not a quota directory (that is, if the quota were set to zero), you would receive the following response:

OK, LQ

"<Current directory>" is not a quota directory.

Total records used = 178.

Records used in this directory = 65.

OK,



Note that although the directory has no quota, information on the records used both by the subtree and by the single directory is still included.

If you give a pathname with the LIST\_QUOTA command, the pathname will appear in the displays above in place of the phrase "<Current directory>". For example:

"MYUFD>REPORTS" is not a quota directory.

If you give the -BRIEF option with the LIST\_QUOTA command, you will receive a one-line summary of quota status for the directory and its subtree. For example:

```
OK, LD REPORTS -BRIEF
Max:      200, Used:      178, Records:      65, REPORTS
OK,
```

In this example, the maximum number of records allowed is 200. The total number of records used for this directory and its subtree is 178. The number of records used by this directory alone is 65.

If you omit the pathname from the command line, the pathname will also be omitted from the one-line summary.

Obtaining Quota and Storage Information with LD: The LD command, explained in Chapter 3, provides quota and storage information on the first line of its display.

For example:

```
OK, LD
<HOUSE>CLOSET (ALL), Records= 47, Quota= 115 / 500
Files= 2.
HANGERS          SHELVES
Segment Directories= 1.
CHAOS.SEG
Directories= 1.
CLOTHING
OK,
```



The number following "Records= " is the number of records used by this directory. The two numbers following "Quota= " are, respectively, the number of records used by the directory and its entire subtree followed by the maximum number of records permitted for use by the directory and its subtree. If the second number is 0, there is no maximum limit other than the limit of the disk. In the example above, the directory CLOSET alone contains 47 records. CLOSET and its subtree contain 115 records. CLOSET and its subtree are permitted to use a maximum of 500 records.

Using LD [pathname] -SIZE: You may wish to learn the number of records in a file or segment directory within a directory. The size of these objects, as well as of directories, is provided by the -SIZE option to the LD command. The display for LD -SIZE for the directory CLOSET is as follows:

```
OK, LD -SIZE
<HOUSE>CLOSET (ALL), Records= 47, Quota= 115 / 500

Files= 2.
    15      dam  HANGERS
    10      sam  SHELVES

Segment Directories= 1.
    21      sseg CHAOS.SEG

Directories= 1.
    68      0 dir  CLOTHING
OK,
```

In this example, the files HANGERS and SHELVES contain, respectively, 15 and 10 records. The segment directory CHAOS.SEG contains 21 records. The subdirectory CLOTHING and its subtree contain 68 records. Note that this column of numbers adds to 114, one less than the 115 listed on the header line as the total content. The additional record stores the directory listing. Very large directories may require more than one record to store their listings.

By using a specific pathname or wildcard pathname, you can request size information on a single object or on a specific group of objects. Additional information on LD appears in the PRIMOS Commands Reference Guide.

Using SIZE: The SIZE command, like the LD -SIZE command, provides the number of records in an existing file, though in a different display. The command format is:

```
SIZE pathname [-NORM]
```



pathname is the name of the object whose size you wish to know. It may be a wildcard name. For example:

OK, SIZE TEXTFILE

13 records in dam file TEXTFILE (13193 words)

OK,

SIZE can report on other file system objects as well. However, for directories, segment directories, and access categories, SIZE returns the number of entries in the object. Hence, the report returned by SIZE depends upon the type of object specified by pathname, as follows:

<u>Object</u>	<u>Report</u>
file	The size of the file in 1024-word records (440-word records if -NORM is specified). The number of words in the file and the file type ("sam file" or "dam file") are also printed.
directory	The number of top-level entries in the directory and the directory type ("pwd UFD" or "acl UFD"). "pwd" = password. The size of the directory listing in words is also reported.
segment directory	The number of entries in the segment directory and the directory type ("sam SEGDIR" or "dam SEGDIR"). The maximum number of entries the segment directory can hold is also reported ("n total"). Multiplying this number by 2 yields the size of the segment directory in words. (For example, "65 total" equals a size of 130 words.)
access category	The number of access pairs (identifier: rights) in the access category.

In all cases, SIZE prints the current pathname, so that you know which object SIZE is looking at when you use wildcards.

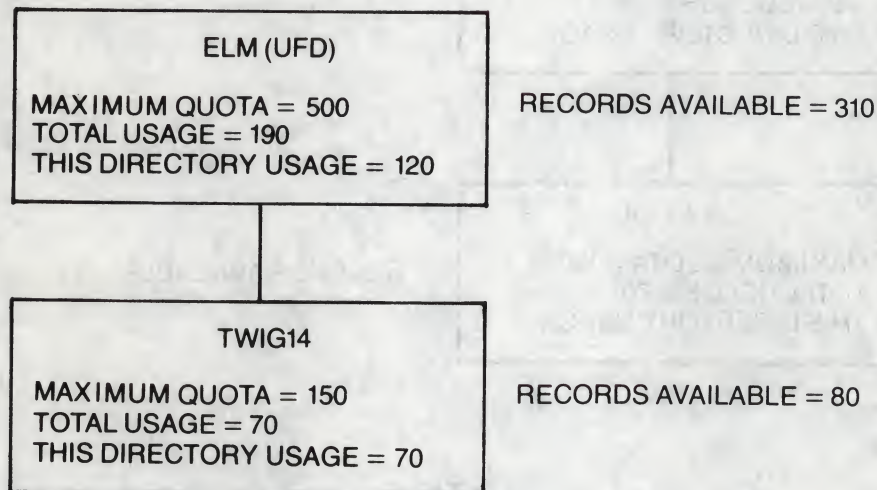
In order to use SIZE in systems that use ACLs, you must have read (R) access to any file or segment directory you wish to size, and list (L) access to any directory you wish to size. You must also have use (U) access to all higher directories in the tree that contains the object.



The following example uses a wildcard to size the entire contents of a UFD, and illustrates the range of reports returned by SIZE:

```
OK, SIZE OURUFD>@@
  3 entries in access cat "OURUFD>COVER.ACAT"
 11 entries in acl UFD    "OURUFD>SUBDIR1" (467 words)
  1 record in sam file    "OURUFD>SMALLFILE" (26 words)
 12 entries in dam SEGDIR "OURUFD>BASE" (65 total)
 24 entries in pwd UFD    "OURUFD>SUBDIR2" (5040 words)
 12 records in dam file   "OURUFD>REPORT" (11816 words)
  4 entries in sam SEGDIR "OURUFD>DATA.SEG" (65 total)
OK,
```

Calculating Storage Availability: To determine how much storage you have left in a directory, you must consider all quotas set on the entire directory tree and also the total current storage used by the entire directory tree. Figures 17-2 and 17-3 illustrate storage availability:



#### Note

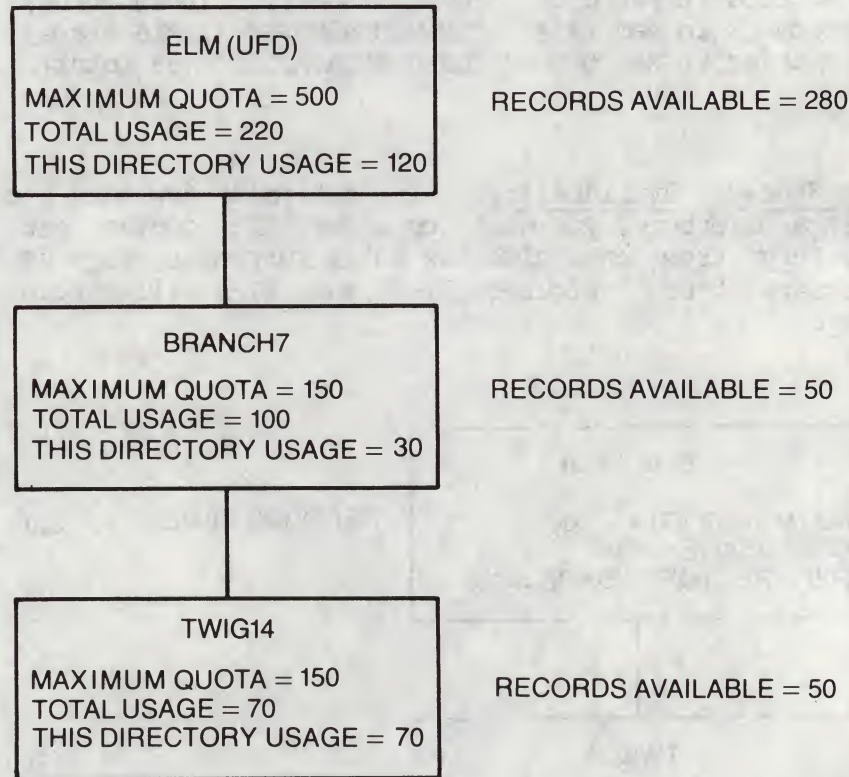
This diagram illustrates the presence of directories only, not the presence of other file system objects in the tree.

Storage Availability on Quota Directories, #1  
Figure 17-2



In Figure 17-2, the subdirectory TWIG14 will allow an additional 80 records in storage (maximum quota of 150 - total usage of 70 = 80). Similarly, the UFD ELM will allow an additional 310 records in storage (maximum quota of 500 - total usage of 190 = 310).

Figure 17-3 modifies the directory tree of Figure 17-2.



Note

This diagram illustrates the presence of directories only, not the presence of other file system objects in the tree.

Storage Availability on Quota Directories, #2  
Figure 17-3

In Figure 17-3, another subdirectory, BRANCH7, exists between ELM and TWIG14. BRANCH7 will allow an additional 50 records in storage (maximum quota of 150 - total usage of 100 = 50). However, the subdirectory TWIG14 in Figure 17-3 will also now allow only 50 additional records in storage. If TWIG14 allowed more than 50 records, then the total usage number in BRANCH7 would exceed its maximum quota of 150 records, which is illegal. (Remember that total usage is the



total records for the subtree below the designated directory plus the records in that directory; for BRANCH7 total usage is  $70 + 30 = 100$ .)

Therefore, the "real" amount of space available on a subdirectory is the smallest of all the differences between the maximum quota and the total usage, calculated for the subdirectory and for each higher directory in its tree.

### Tips on Using Quotas

The "Disk-Full" Condition: The system administrator can assign UFD quotas whose sum exceeds the capacity of the disk. This capability assumes that not all users will be using their full storage allotment at the same time. In effect, users "share" part of their space, which provides more efficient use of the disk. If, however, you attempt to store an object and the disk is completely full, you will get the message "The disk is full." You will not be able to store the object until someone on the system has deleted at least as many records as your object needs.

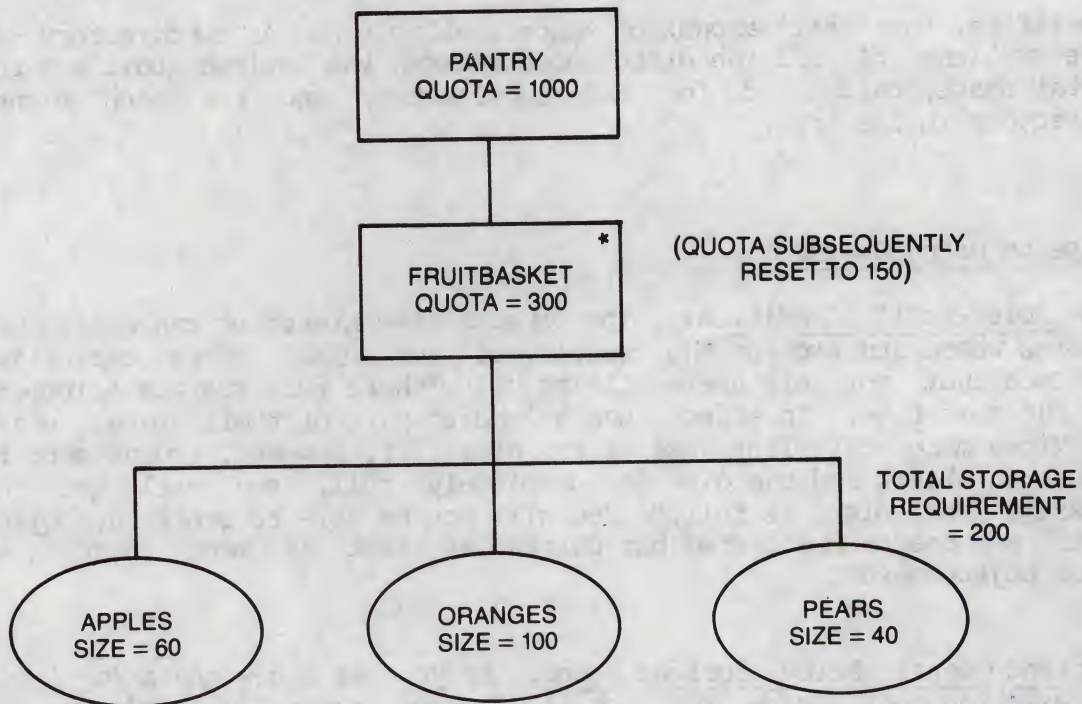
Setting Quotas Below Current Usage: If you set a new quota (or lower an existing one) below the actual storage currently used by the directory, PRIMOS will give you the message "QUOTA set below current usage." With a directory in this state, you can delete objects or reduce the size of existing objects. You cannot, however, enlarge existing objects or create new objects in the directory until you reduce the current number of records to a level below that of the new quota. The exception is ACLs, which you may still be able to create and enlarge as well as reduce or delete, until you require an additional record from the file system.

For example, suppose you have the files APPLES, ORANGES, and PEARS, in the subdirectory FRUITBASKET. The files contain 60, 100, and 40 records, respectively; the quota on the directory is 300. Figure 17-4 illustrates the directory tree. You now reset the quota to 150 records using the command:

```
OK, SQ PANTRY>FRUITBASKET -MAX 150
QUOTA set below current usage. FRUITBASKET (set_quota)
OK,
```

With this new quota set on the directory, you can, say, delete APPLES or reduce ORANGES to 50 records. You cannot increase PEARS to 60 records or create a new file called BANANAS until you either reduce the total number of records in FRUITBASKET to below the current quota of 150, or raise the quota to a level above the current total storage of 200 records.





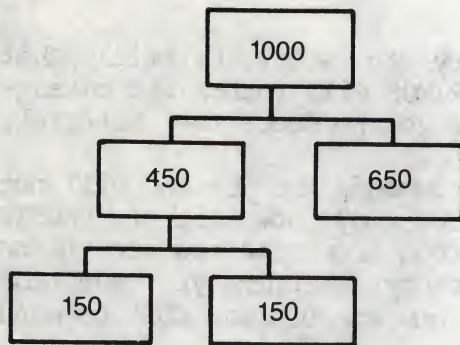
\* = attach point

Quota Usage  
Figure 17-4

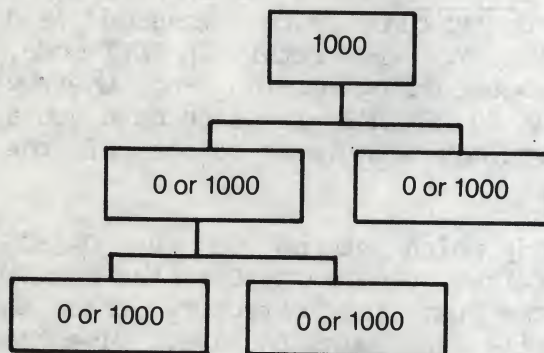
Setting Useful Quotas: You may set subdirectory quotas that exceed quotas on higher level directories (including the UFD). However, you may not actually use more total space in a directory than is allotted to it or to any higher directory in its immediate tree.

You should try to set quotas that accurately reflect the number of records that you expect to use. For example, consider the directory trees in Figure 17-5. The numbers indicate quotas set on the directories.

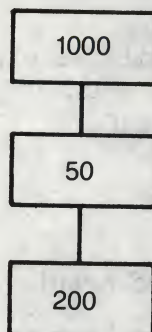




- A. Good use of quotas  
(Quotas decrease as  
directory levels  
deepen.)



- B. Good use of quotas  
(Since 0 means no  
limit, all sub-UFDs  
are, in effect, using  
the top-level quota.)



- C. Poor use of quotas  
(Middle quota is not  
realistic with respect  
to the lowest quota.)

Setting Useful Quotas  
Figure 17-5

Diagrams A and B reflect good usage of quotas. In tree A, quotas decrease in size as directory levels deepen. In tree B, with either no quotas (0 = no limit) or with the UFD quota of 1000 also set on all subdirectories, all subdirectories share available space equally. Tree C, however, illustrates an unrealistic usage of quotas at the third directory level. Suppose you had a file **HUNDRED**, 100 records long; you could save it only in C's top level UFD, which admits 1000 records. Neither of the two other subdirectories could accommodate **HUNDRED**. Although the quota on the lowest sub-UFD (200) is sufficient, a higher-level quota (that is, 50) is too small. In effect, the quota on the lowest subdirectory may as well be 50, too.



Recovering from Quota Overloads: If you try to store material that will cause a quota to be exceeded, PRIMOS will return the message "Maximum quota exceeded" and will not allow you to store the material.

If, for example, you are using the COPY command, the command will not execute, and you will return to PRIMOS. You may now either DELETE enough material to provide sufficient room for the new records or increase the quota on the problematic directory (assuming you have the appropriate access privileges). You may now execute the COPY command again.

Quota overloads that occur during EDITing sessions pose an additional complication. If the command "FILE filename" would result in a quota overload, PRIMOS will return the message "Maximum quota exceeded" and will not allow you to save the file. You will remain in EDIT mode. You will probably not wish to cancel your editing session and thereby lose your work in order to return to PRIMOS level to make quota adjustments. To save your material, you may be able to do one of the following:

1. Give the Editor's PAUSE command, which returns you to PRIMOS level without ending your Editor session. Use the DELETE command to remove enough objects from the directory tree to provide room for the new file you wish to save. Give the command START to resume your editing session, and save your work.

In the following example, TOOBIG is the file we wish to save in the current directory. EXTRANEIOUS is another file, large enough to provide the space needed by TOOBIG.

```
OK, ED
INPUT
-----
[contents of TOOBIG entered here]
-----
(CR)
EDIT
FILE TOOBIG
Maximum quota exceeded. TOOBIG (OPENR)
?
PAUSE
OK, DELETE EXTRANEIOUS
OK, START
FILE TOOBIG
OK,
```

Following the PAUSE command, you may, if you wish, use the COPY command to copy the objects you intend to delete into another directory before you delete them.



Note that after you give the `START` command and the customary carriage return, PRIMOS returns you to your editing session. It does not, however, give you any prompt to signify this.

2. Give the `PAUSE` command as in #1. If the directory with the problematic quota is a directory to which you have protect access rights (for ACL systems) or owner rights (for directory password systems), use the `SET_QUOTA` command to reset the quota to a higher number.

In the following example, `TOOLARGE` is the file we wish to save in the current directory `WRHAUS>STORAGEBLOC`. `STORAGEBLOC` has a quota of 100 records and `TOOLARGE` is 110 records long. `WRHAUS` has a quota of zero (that is, no limitation). The directories are empty, and we have protect access to the entire tree:

```
OK, ED
INPUT
-----
----- [contents of TOOLARGE entered here]
-----
(CR)
EDIT
FILE TOOLARGE
Maximum quota exceeded. TOOLARGE (OPENR)
?
PAUSE
OK, SQ WRHAUS>STORAGEBLOC -MAX 200
OK, START
FILE TOOLARGE
OK,
```

3. File your work in another directory to which you have add rights (for ACL systems) or owner rights (for directory password systems). Since you will now return to PRIMOS level, either `DELETE` materials in the tree or enlarge the quota (if you can control the quota on the problematic directory). Finally, `COPY` the file from its temporary storage into the proper directory.

If none of these solutions are possible or desirable, see your administrator.



The first part of the document discusses the importance of maintaining accurate records of all transactions. It emphasizes that proper record-keeping is essential for the integrity of the financial system and for the ability to detect and prevent fraud.

In the second part, the document outlines the specific procedures for recording transactions. It details the steps involved in the accounting cycle, from identifying the transaction to posting it to the appropriate ledger account. It also discusses the importance of double-checking entries to ensure accuracy.

The third part of the document addresses the role of internal controls in the accounting process. It explains how internal controls help to minimize the risk of errors and fraud by establishing a system of checks and balances. It also discusses the importance of regular audits to ensure that the internal controls are effective.

The fourth part of the document discusses the importance of transparency and accountability in the accounting process. It emphasizes that all transactions should be recorded in a clear and concise manner, and that the results of the accounting process should be made available to all stakeholders.

The fifth part of the document discusses the importance of staying up-to-date on changes in accounting standards and regulations. It emphasizes that the accounting profession is constantly evolving, and that accountants must stay current in their knowledge and skills to ensure that they are providing accurate and reliable information.

The sixth part of the document discusses the importance of ethical behavior in the accounting profession. It emphasizes that accountants have a responsibility to act honestly and fairly, and to avoid any conflicts of interest. It also discusses the importance of maintaining confidentiality and protecting the privacy of the information that is entrusted to them.

The seventh part of the document discusses the importance of communication in the accounting process. It emphasizes that accountants must be able to communicate effectively with all stakeholders, including management, investors, and the public.



# 18

## Command Line Features

### INTRODUCTION

PRIMOS supports many features that allow you to control command processing with precision and ease. In this section we shall discuss these facilities, which allow you to:

- Use iteration to tell PRIMOS to repeat a command, substituting a new file system object each time.
- Use wildcard symbols to tell PRIMOS to execute a command on a certain group of objects without listing all their names individually.
- Use the treewalking feature to tell PRIMOS to search through directory levels and execute the specified command on the appropriate objects.
- Use the name generation feature to avoid repeating long objectnames by telling PRIMOS to substitute the full names for generation symbols.

### ITERATION

There are occasions when you want to repeat a command several times, specifying a new file (or other file system object) each time.



For example:

```
FTN A -64V -XREF
FTN B -64V -XREF
FTN C -64V -XREF
```

You can do this more easily by giving one command which contains all three filenames enclosed in parentheses. For example:

```
FTN (A B C) -64V -XREF
```

When you do this, the parenthesized list you create is called an iteration list. PRIMOS will execute the command once for each member of the list, just as if you had typed the commands separately.

#### How to Create Iteration Lists

The simplest method of creating an iteration list is that shown above--enclosing a list of items, separated from each other by one or more blanks, within a set of parentheses. However, iteration allows more flexibility than that. Additional capabilities follow.

Like other items on the command line, items in iteration lists may be separated by commas, instead of by blanks. For example:

```
FTN (A,B,C) -64V -XREF
```

This command is identical in operation to the command in the preceding example.

Multiple Iteration Lists: You may replace as many arguments as you like with iteration lists. Just remember to leave a blank space (or comma) between the lists, as you would between the arguments themselves. For example:

```
COPY (A B C) (D E F)
```

The command above copies files A, B, and C, renaming them D, E, and F, respectively. The iteration list (A B C) is the first argument, and the iteration list (D E F) is the second.

When an iteration list runs out of arguments, a null string is substituted. For example, the command:

```
CMPF (L M N) (O P) (Q R S)
```

results in the commands:

```
CMPF L O Q
CMPF M P R
CMPF N S
```



Iteration Lists as Parts of Arguments: The previous examples show iteration lists being used as complete arguments. However, the iteration list may be only part of the argument. The rule here is, anything not separated by a blank or a comma is a single argument. Hence, you can give a command such as:

```
COPY MYDIR>(A B C) HERDIR>(D E F)
```

This command copies the files A, B, and C from the directory MYDIR into the directory HERDIR, naming them D, E, and F.

Cross-product Iteration: Two—but no more than two—iteration lists can be used within a single argument. When this is done, it causes cross-product iteration. That is, each item in the first iteration list is paired with each item in the second iteration list, and the command is executed once for each object that results. For example, the command:

```
DELETE (A B C) (D E F)
```

deletes the following nine files:

```
AD
AE
AF
BD
BE
BF
CD
CE
CF
```

Similarly, the command:

```
DELETE A(B,C,D).(LIST,BIN)
```

deletes the six files:

```
AB.LIST
AB.BIN
AC.LIST
AC.BIN
AD.LIST
AD.BIN
```



WILDCARDING

Wildcarding allows you to specify groups of file system objects on which a command can act. For example, the command:

```
FIN @@.FIN
```

compiles all files in the current directory whose names end in .FIN. The command:

```
LD BEECH>A@@
```

lists all file system objects in the directory BEECH whose names begin with the letter A.

Wildcarding is specified by using a wildcard name as an argument to a command. A wildcard name is a pathname in which the final element (or the only element, if the name is a simple filename) contains one or more of the wild characters shown in Table 18-1.

Table 18-1  
Wild Characters

Character	Function
@@	Replaces any number of characters in any number of components within a file or directory name.
@	Replaces any number of characters within one component of a filename or directory name. Stops matching at the period (.) that separates a name and its suffix.
+	Replaces a single character, except a period(.).
^	Negation character. The negation character must be the first character in the wildcard name. A wildcard name that begins with ^ matches all names that <u>don't</u> match the rest of the wildcard name.

Only one argument per command can contain wildcard names. That argument can contain either a single wildcard name (as in the examples above), or a single iteration list containing any number of wildcard names, as in the following examples:

```
DELETE (A@@ B@@)
```

```
DELETE @@(.BIN .LIST)
```



Wildcard Matching

When a command containing a wildcard name is given, the command processor searches the specified directory for all file object names that "match" the given wildcard name. A file object's name is said to match a wildcard name if:

- it has the same number of components as the wildcard name, and
- it contains the same literal characters as the wildcard name contains, in the same relative positions.

The three wildcard symbols +, @, and @@ differ only in the number of characters they can match. The + can only match a single character (except a period). The @ matches any number of characters in a single component of a name, but it cannot cross the period (.) that separates components. The @@ matches any number of characters in any number of components. Thus, a wildcard name consisting only of the symbol @ would match all single-component names in the directory, while a name consisting of the symbol @@ would match all names in the directory, no matter how many components they contained.

The selection order for wildcard matching is unpredictable.

Examples of Wildcard Names

Suppose that the current directory contains the file system objects:

BARR1.COBOL	BARR1.SEG	BARR2.COBOL	BARR2.SEG
CLR.CPL	EDD.COMO	EDD.COMO.OLD	EDD.CPL
EDD.CPL.OLD	FILL	SCROLL	SKILL

The wildcard name @ matches all one-component names:

FILL	SCROLL	SKILL
------	--------	-------

The wildcard name S@ matches all one-component names that begin with S:

SCROLL	SKILL
--------	-------

The wildcard name @.@ matches all two-component names:

BARR1.COBOL	BARR1.SEG	BARR2.COBOL	BARR2.SEG
CLR.CPL	EDD.COMO	EDD.CPL	

The wildcard name @.SEG matches all two-component names which end in .SEG:

BARR1.SEG	BARR2.SEG
-----------	-----------



The wildcard name @.@.@ matches all three-component names:

EDD.COMO.OLD      EDD.CPL.OLD

The wildcard name BARR+.COBOL matches:

BARR1.COBOLE      BARR2.COBOLE

The wildcard name BARR+.@ matches:

BARR1.COBOLE      BARR1.SEG      BARR2.COBOLE      BARR2.SEG

The wildcard name @@L matches all names, of any number of components, that end in L:

BARR1.COBOLE      BARR2.COBOLE      CLR.CPL      FILL  
EDD.CPL      SCROLL      SKILL

The wildcard name @@ matches every name in a directory.

### Inverted Matching

Any wildcard name may be preceded by a caret (^), which serves as an inverted match character. Such a wildcard name matches every object whose name does not match the wildcard name. In the example above, the wildcard name ^@@L matches all names that do not end in L:

BARR1.SEG      BARR2.SEG      EDD.COMO      EDD.COMO.OLD  
EDD.CPL.OLD

### Wildcard Options

The effect of wildcard names can be altered by specifying one or more of the wildcard options shown in Table 18-2. If you use wildcard options, they may appear anywhere after the command in the command line. Most PRIMOS commands allow the use of wildcards.



Table 18-2  
Wildcard Options

Option	Abbreviation	Objects Selected
-FILE	-FILE	SAM or DAM files
-DIRECTORY	-DIR	MFDs, UFDs, and Sub-UFDs
-SEGMENT_DIRECTORY	-SEGDIR	SAM or DAM segment directories
-ACCESS_CATEGORY	-ACAT	Access categories
-BEFORE date.time	-BF date.time	An object will be selected only if it was last modified on or before <u>date.time</u> .
-AFTER date.time	-AF date.time	An object will be selected only if it was last modified on or after <u>date.time</u> .
-VERIFY	-VFY	PRIMOS will request verification of each object before executing the command on that object.
-NO_VERIFY	-NVFY	PRIMOS will not request verification even for a command that usually requires it, such as DELETE. (-NO_VERIFY is the normal default.)

Wildcard options may be used to:

- Select only file system objects that are of a designated type or types.
- Select only file system objects that were last modified before or after a particular date.
- Enable or disable verification.



Type-designation Options: The type-designation options are the following:

<u>Type</u>	<u>Abbreviation</u>
-FILE	
-DIRECTORY	-DIR
-SEGMENT_DIRECTORY	-SEGDIR
-ACCESS_CATEGORY	-ACAT

If one or more of these options is given in a command line containing a wildcard name, the command will execute only for objects that have those designated types. For example, the command:

```
LD A@@ -FILE
```

lists all the files beginning with A in the current directory. The command:

```
LD A@@ -FILE -ACCESS_CATEGORY
```

lists all the files and all the access categories beginning with A in the directory. If none of the type-specification options are given, all file object types (files, directories, segment directories, and access categories) are matched.

Date-selection Options: There are two date-selection options:

<u>Option</u>	<u>Abbreviation</u>
-BEFORE date.time	-BF date.time
-AFTER date.time	-AF date.time

Giving these options causes the command processor to match only objects that were last modified before or after a given date and time.

There are four full formats for date.time:

```
mo/dd/yy.hh:mi:ss
```

```
yy-mo-dd.hh:mi:ss
```

```
'dd mon yy.hh:mi:ss'
```

```
'dd mon yy hh:mi:ss'
```



mon is the first three characters of the month (JAN, FEB, MAR, etc.).  
All other symbols are one-digit or two-digit numbers, representing the following:

<u>Symbol</u>	<u>Meaning</u>
yy	year (19yy)
mo	month (January = 1 or 01, and so on)
dd	day (1 or 01 to 31)
hh	hour (24 hour designation)
mi	minute
ss	second

Not all items in the full date.time formats are required. The following defaults are assumed when items are omitted:

<u>Omitted Item</u>	<u>Default Value</u>
mo/dd/yy yy-mo-dd dd mon yy	today's date
yy	today's year
hh:mm:ss	00:00:00 (the beginning of the specified day)
mm	00
ss	00

Examples of valid dates and times:

```
-AFTER 03/07/82.13:23:09
-BEFORE 82-3-7.13:23:9
-AFTER '07 MAR 82 13:23:9'
-AFTER 11:15:35 (after 11:15 AM, plus 35 seconds, today's date)
-BEFORE 1/14/82 (before January 14,1982)
-AFTER 2-22 (after midnight as 21 Feb turns to 22 Feb)
-BEFORE 16 (before 4 PM, today's date)
-AFTER 3. (after the beginning of the 3rd day, current month
and year)
-BEFORE '3 MAR.11' (before 11 AM, on 3 March, current year)
-AFTER 0 (everything after midnight of last night)
```



The presence or absence of the period separator can determine how a date and time are interpreted. For example:

<u>Designation</u>	<u>Meaning</u>
-BEFORE 10	Before 10 AM, today's date
-BEFORE 10.	Before the 10th day, current month and year

If both a -BEFORE and an -AFTER option are given, an object must satisfy both options in order to be selected. For example, the command:

```
LD @@ -SEGDIR -AFTER 12/15/81 -BEFORE 3/1/82
```

lists only segment directories that were last modified between December 15, 1981 and March 1, 1982.

Verification Options: The verification options are -VERIFY and -NOVERIFY. These options differ from other wildcard options in that they do not control the actual matching of objects. Rather, they enable or disable verification for the commands that act on the chosen objects, as explained below.

How Verification Works: Some commands, such as the DELETE command, cause PRIMOS to take actions which are irrevocable once performed. When such a command is given with a wildcard name, PRIMOS asks the user for verification before each execution of the command. This ensures that the user really does want the action performed on that particular object.

For example, assume that you have the files BAT, CAT and RAT in the current directory and that you wish to delete BAT and RAT:

```
OK, DELETE @@AT
(std$cp) Verify wildcard selections for "@@AT":
"BAT"? YES
"CAT"? NO
"RAT"? YES
OK,
```

The response "YES" or "Y" to the verification request will cause the command to be executed for the name shown in the request as soon as all names have been verified. The response "NO", "N", or (CR) causes execution to be skipped for the name shown. The response "NEXT" cancels execution for all names, even those previously verified with "YES". This cancellation does not affect any iteration or treewalking that is in progress. Hitting the BREAK key ends all processing of the command line, and returns the user to PRIMOS command level.



Verification may be disabled for a command that normally requests verification by using the `-NO_VERIFY` option. Verification may be enabled for a command that normally does not require it by specifying the `-VERIFY` option.

### TREEWALKING

Wildcarding allows PRIMOS commands to act on a group of file system objects located within a single directory. Treewalking takes this convenience one step farther, and allows a command to act on designated objects within a directory tree: that is, a directory, its subdirectories, their subdirectories, etc.

Treewalking is specified by using wildcard characters in some intermediate position within a pathname. The wildcard characters cannot be in the first position of the pathname. (A treewalking pathname may also have a simple wildcard name in the final position of the pathname.)

For example, Figure 18-1 represents a sample directory tree, stemming from the directory ORCHARD.

If we attached to directory ORCHARD and gave the command:

```
LD @@ -DIR
```

the LD comand would list all the directories contained in ORCHARD: APPLETREE, PEACHTREE, and PEARTREE. (The command could also be given as `LD -DIR`, since @@ is assumed when no pathname follows the LD command.) The display would look like this:

```
<DISK>ORCHARD (ALL), Records= 1, Quota= 18 / 0
```

```
Directories= 3.
```

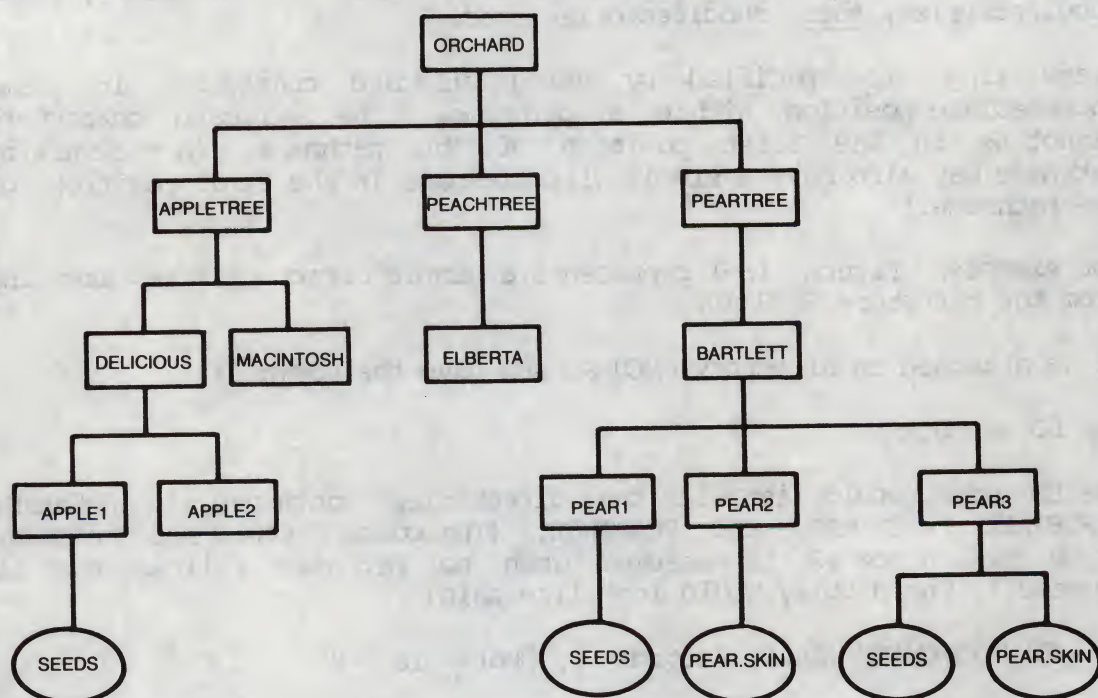
```
APPLETREE    PEACHTREE    PEARTREE
```

However, if we gave the command:

```
LD ORCHARD>@@>@@ -DIR
```

we would receive information on all subdirectories subordinate to ORCHARD, from APPLETREE to PEAR3. The display might look like Figure 18-2.





Sample Directory Tree  
Figure 18-1



```

OK, LD ORCHARD>@@>@@ -DIR
<DISK>ORCHARD>APPLETREE (ALL), Records= 1, Quota= 6 / 0
Directories= 2.
DELICIOUS          MACINTOSH
<DISK>ORCHARD>APPLETREE>DELICIOUS (ALL), Records= 1, Quota= 4 / 0
Directories= 2.
APPLE1             APPLE2
<DISK>ORCHARD>APPLETREE>DELICIOUS>APPLE1 (ALL), Records= 2, Quota= 2 / 0
<DISK>ORCHARD>APPLETREE>DELICIOUS>APPLE2 (ALL), Records= 1, Quota= 1 / 0
<DISK>ORCHARD>APPLETREE>MACINTOSH (ALL), Records= 1, Quota= 1 / 0
<DISK>ORCHARD>PEACHTREE (ALL), Records= 1, Quota= 2 / 0
Directories= 1.
ELBERTA
<DISK>ORCHARD>PEACHTREE>ELBERTA (ALL), Records= 1, Quota= 1 / 0
<DISK>ORCHARD>PEARTREE (ALL), Records= 1, Quota= 9 / 0
Directories= 1.
BARTLETT
<DISK>ORCHARD>PEARTREE>BARTLETT (ALL), Records= 1, Quota= 8 / 0
Directories= 3.
PEAR1              PEAR2              PEAR3
<DISK>ORCHARD>PEARTREE>BARTLETT>PEAR1 (ALL), Records= 2, Quota= 2 / 0
<DISK>ORCHARD>PEARTREE>BARTLETT>PEAR2 (ALL), Records= 2, Quota= 2 / 0
<DISK>ORCHARD>PEARTREE>BARTLETT>PEAR3 (ALL), Records= 3, Quota= 3 / 0
OK,

```

Sample Treewalking Terminal Display  
Figure 18-2



Further Examples of Treewalking

We could also list all the directories in the subtree headed by directory APPLETREE. We would do this with the command:

```
LD ORCHARD>APPLETREE>@@>@@ -DIR
```

We could locate all files named SEEDS in our sample directory tree by giving the command:

```
LD ORCHARD>@@>SEEDS
```

We could find all files ending in .SKIN in the subtree headed by directory PEARTREE by giving the command:

```
LD ORCHARD>PEARTREE>@@>@@.SKIN
```

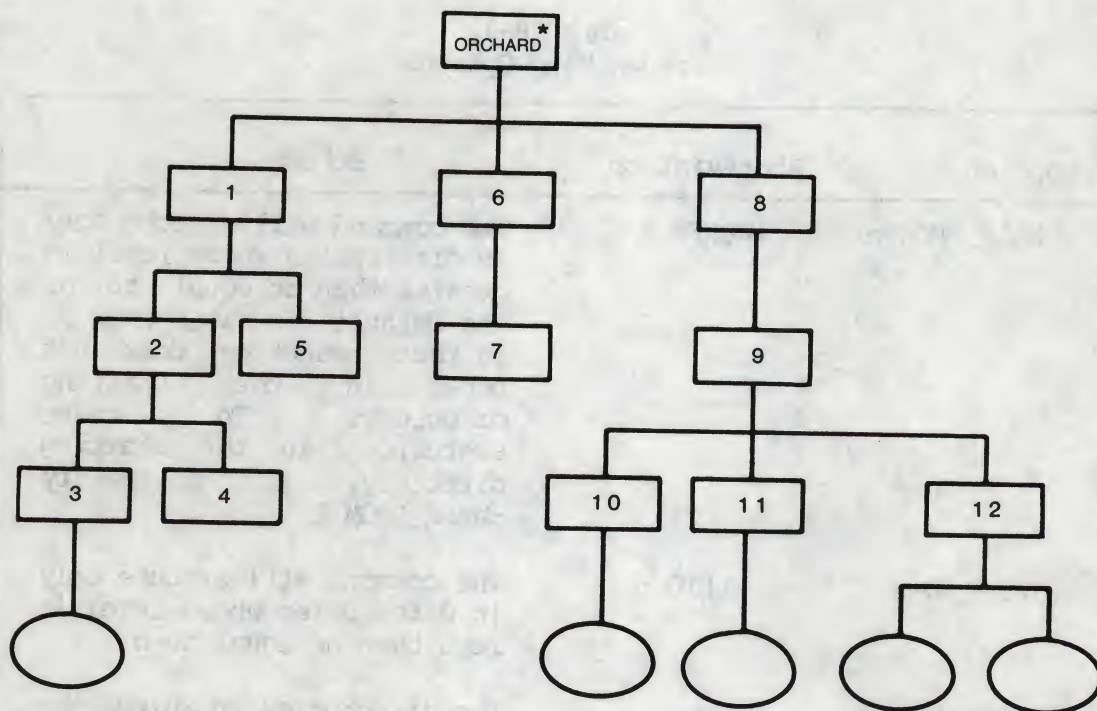
These examples have used the common wildcard character, @@. However, any wildcard characters can be used in a treewalk name. When used in treewalk names, they have the same meanings that they have in wildcard names.

How Treewalking Happens

When a command that contains a treewalk name is given, the command processor searches all directories subordinate to the specified starting directory for file system objects that match the given treewalk pathname. (Note that the starting directory itself is not searched. To cause the starting directory to be searched, you must specify the "WALK\_FROM 1" option, explained below.)

The diagram in Figure 18-3 illustrates the way that PRIMOS proceeds through directories vertically in a standard treewalk. The horizontal order of visitation within a single directory cannot be predicted. Hence, in Figure 18-3 the sample horizontal order corresponds to APPLETREE, then PEACHTREE, then PEARTREE, but this order could just as well be different.





\* = attach point

Command = LD ORCHARD>@@>@@

Order of Visiting Directories  
in Sample Standard Treewalk

Figure 18-3



To change the order of the search, or the number of directories visited, you can use the treewalking options shown below.

### Treewalking Options

The treewalking options are explained in Table 18-3.

Table 18-3  
Treewalking Options

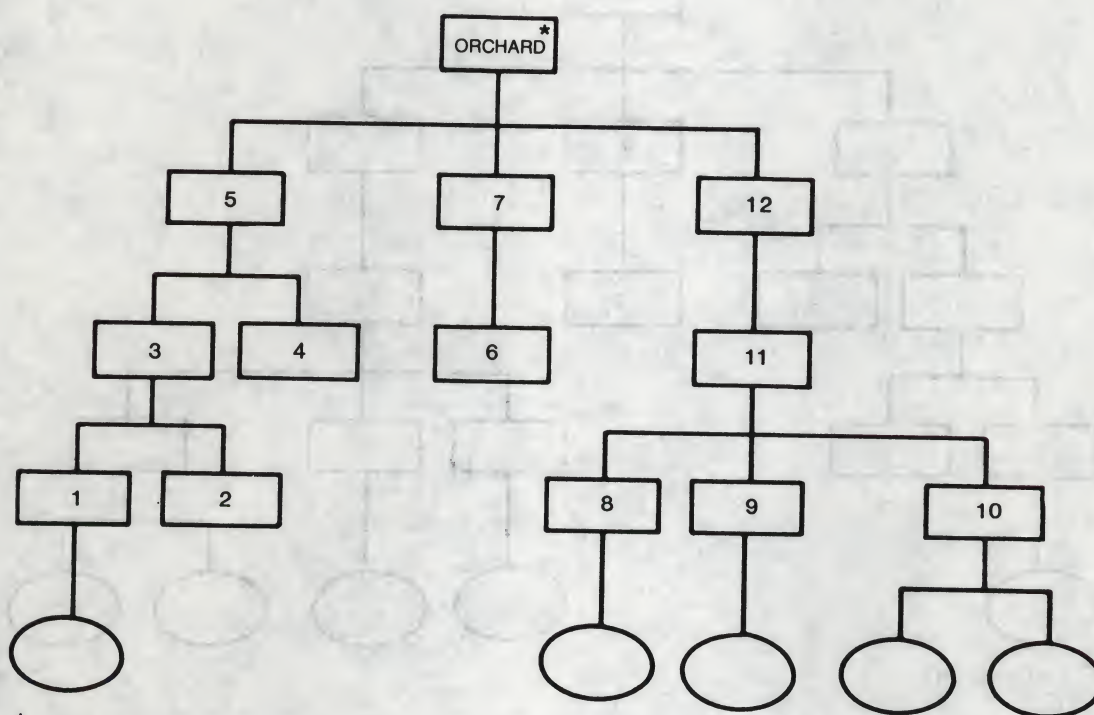
Option	Abbreviation	Effect
-WALK_FROM n	-WLKFM n	The command will execute only in directories whose level is greater than or equal to <u>n</u> . The default is -WALK_FROM 2, so that execution does not occur in the starting directory. To cause execution in the starting directory, specify -WALK_FROM 1.
-WALK_TO n	-WLKTO n	The command will execute only in directories whose level is less than or equal to <u>n</u> .
-BOTTOM_UP	-BOTUP	The directories in which the command executes will be visited in "bottom-up" order, starting with the largest level number and going to the smallest level number. (The default is to start at the smallest level number and work "down" to the largest level number.)



Examples of Treewalking Using Options

1. Figure 18-4 illustrates the order of visiting directories using the command for a bottom-up treewalk in the sample tree ORCHARD. The command is:

```
LD ORCHARD>@@>@@ -BOTUP
```



\* = attach point

Command = LD ORCHARD>@@>@@ -BOTUP

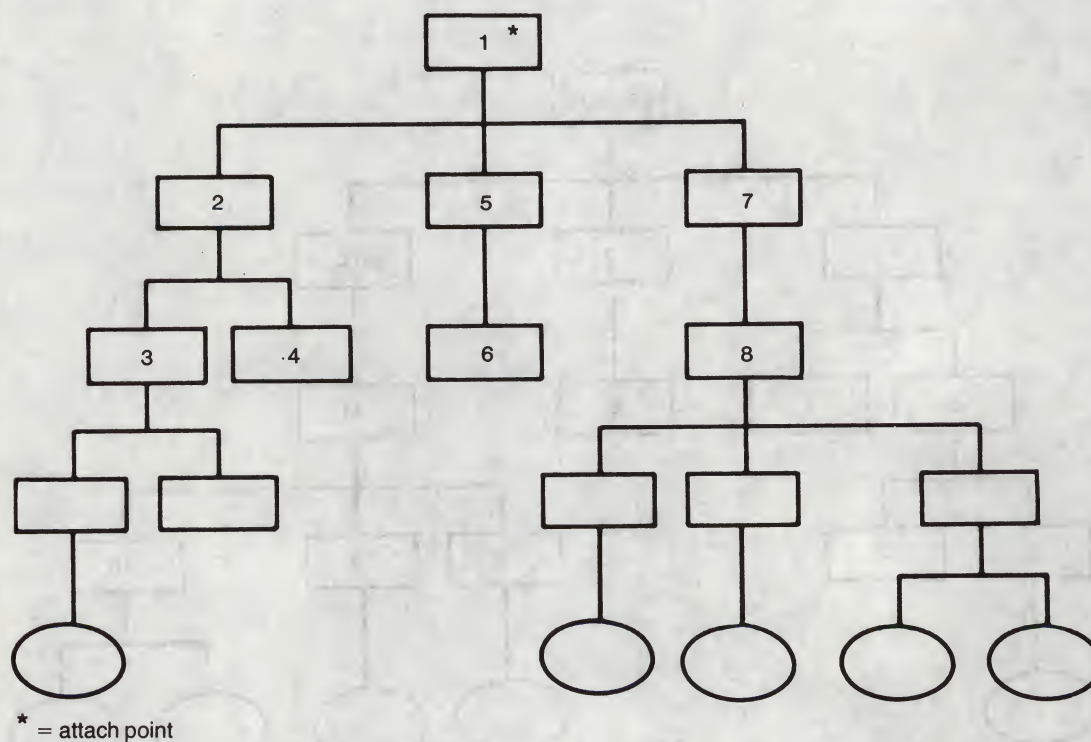
Order of Visiting Directories  
in Sample Bottom-up Treewalk

Figure 18-4



2. Figure 18-5 illustrates the order of visiting directories using the LD command for a treewalk which includes the current directory but not the lowest directory level. The sample directory tree is, again, ORCHARD. The command is:

```
LD ORCHARD>@>@ -WALK_FROM 1 -WALK_TO 3
```



Command = LD ORCHARD>@>@ -WALK\_FROM 1 -WALK\_TO 3

Effect of Specifying Levels  
to Treewalking Command

Figure 18-5



NAME GENERATION

Some commands require the use of several objectnames. If the objectnames are nearly identical, and if they are long, typing them in full may be a nuisance.

Also, there may be times when you want to create a group of names that will parallel a group of wildcard names (for example, when copying and renaming a group of files).

In these cases, the use of name generation can be a convenient shortcut. For example, a merge could be done with the command:

```
MRGF FILE.1 =.2 =.3 -OUTF =.4 .
```

This command compares FILE.1 against FILE.2 and FILE.3, and creates FILE.4 as a merged output file.

How to Use Name Generation

Name generation requires at least two pathnames:

- the source pathname, from which to create new names
- one or more pathnames containing generation patterns in their objectname portions

Source Pathname: The source pathname is usually the first pathname in the command line. That is, it usually forms the first argument to the command.

For example, in the command line:

```
CMPF DIR>NAMES.OLD DIR>NAMES.NEW
```

the source pathname is DIR>NAMES.OLD.

The exceptions to this rule are the RESUME and SEG commands. When these commands are given, the source pathname is the second argument in the command line.

For example, in the command line:

```
RESUME MYPROGRAM DIR>ARGUMENT1 DIR>ARGUMENT2
```

the source pathname is DIR>ARGUMENT1.



Generation Patterns: Name generation patterns are composed of name generation characters (usually = and ==), and literal strings of characters. One equal sign (=) copies a single component. The double equal sign (==) copies as many components as can be copied without adding components to the name. This happens because the command processor, unless told otherwise, assumes that the number of components in a generated name will be equal to or less than the number in the original name. For this reason, only one double equal sign can appear in a name generation pattern. (If two were to appear, the command processor would have no way of knowing how to allocate components between them.)

A literal string of characters replaces a component from the source name.

A summary of name generation symbols and their effects is shown in Table 18-4.

Table 18-4  
Name Generation Symbols and Their Effects

Command	Effect
=	Copies a single component from the source name to the generated name. (Parallels the wild character @ for copying.)
==	Copies one or more components from the source name to the generated name. (Parallels the wild character @@ for copying, except for such substitutions, additions, or deletions as are specified.)
^=	Skips over a single component from the source name without copying it to the generated name.
^==	Skips over one or more components of the source name without copying them to the generated name.
literal-string	Replaces a component from the source name with the component given by <u>literal-string</u> .
+literal-string	Adds to the generated name the component given by <u>literal-string</u> .



Examples of Name Generation

<u>Source Name</u>	<u>Generation Pattern</u>	<u>Generated Name</u>	<u>Comments</u>
A.B.C.D	==.X	A.B.C.X	Three components are copied to create a 4-component name.
A.B.C.D	==.X.Y	A.B.X.Y	The first two components are copied.
A.B.C.D	X.==.X	X.B.C.X	The middle two components are copied.

However:

A.B.C.D	X.=	X.B	A single equal sign copies only one component.
A.B.C.D	=.X.Y.Z.=		An error message is sent, since the pattern would require 5 components and the source name only contains 4.

Note

Name generation patterns can be used only in the objectname portion (that is, the final position) of the pathname. They cannot be used in any other portion. Thus, you cannot say:

COPY A>LONGNAME>B X>=>Y

Adding Components

To add a component to a generated name, precede the new characters with a plus sign (+). For example:

<u>Source Name</u>	<u>Generation Pattern</u>	<u>Generated Name</u>
A.B	==.+C	A.B.C
A.B	==.+C.=	A.C.B
A.B	+C.==	C.A.B



The exception to this rule occurs when you are adding a component to the end of a name, and have specified each preceding component explicitly. In this case, you do not need to use the plus sign. Simply type in the new component literally. For example:

A.B                    =.=.C                    A.B.C

### Deleting Components

To delete one or more components, precede the equal or double equal sign with a NOT sign (^).

#### Examples:

A.B.C	=.^==	A
A.B.C	^=.=	B.C
A.B.C	=.^=.=	A.C
A.B.C	^==.=	C

### Combining Additions, Substitutions, and Deletions

Additions, substitutions, and deletions may all be used in a single name generation pattern. For example:

A.B.C.D.E.F            =.X.+Y.=.^=.=            A.X.Y.C.E

However, remember that only one double equal sign, with or without a NOT sign, may appear in the pattern.

### COMBINING COMMAND LINE FEATURES

#### Combining Iteration with Other Features

Wildcards, treewalk patterns, and name generation patterns, as well as abbreviations, variables, and function calls, can all be used within iteration lists. For example, the following are all legal commands:

```
DELETE (A@@ B@@ C@@)
COPY (@.LIST @.BIN) ARCHIV>(=.OLDLIST =.OLDBIN)
SPOOL (%.YESTERDAY% %.TODAY% %.TOMORROW%)
```



Wildcard Characters and Name Generation

If a source pathname includes a wildcard name, the generated names will match whatever names are produced by the wildcarding process. For example, a copy command might read:

```
COPY ALPHA>*.LIST ARCHIV>*.OLDLIST
```

This command copies all listing files in the directory ALPHA into the directory ARCHIV. It changes the suffix on each from "LIST" to "OLDLIST". It might, for example, generate the three commands:

```
COPY ALPHA>A.LIST ARCHIV>A.OLDLIST  
COPY ALPHA>B.LIST ARCHIV>B.OLDLIST  
COPY ALPHA>C.LIST ARCHIV>C.OLDLIST
```



1. The first part of the report is a summary of the work done during the year.

2. The second part of the report is a detailed account of the work done during the year.

3. The third part of the report is a summary of the work done during the year.

4. The fourth part of the report is a detailed account of the work done during the year.

5. The fifth part of the report is a summary of the work done during the year.



# 19

## Command Line Processing

### INTRODUCTION

If you are using the many new features of the Primos command processor frequently, you may find it useful to know the order in which they are processed and the interactions between them. These interactions contribute to the power of the PRIMOS command line. This chapter discusses these features by presenting a chronology of the processing of a command line. The chapter concludes with an example that diagrams the processing of a complex command line.

Command line processing proceeds in the following order:

1. Abbreviation Expansion
2. Syntax Suppression
3. Multiple Command Processing
4. Variable and Function Evaluation
5. Iteration
6. Treewalking
7. Wildcarding
8. Name Generation
9. Execution



ABBREVIATION EXPANSION

If the user has an ABBREV file enabled, the first thing that happens to the command line is that abbreviations are expanded. (Remember, the System Administrator can disable all abbreviation expansion on his system.)

Abbreviation Interaction with Iteration Lists & Command Functions

The abbreviation processor treats a command function or simple iteration list as a single item (or token). This affects only functions or iteration lists that require abbreviation parameters or that are used as arguments for an abbreviation parameter. For example, if:

A is the abbreviation for `*>SUBDIR>%1%`

then:

A (B C D)

expands to:

`*>SUBDIR>(B C D)`

In this example, the list (B C D) is treated as a single token and is assigned as the value of parameter 1 of abbreviation A.

Similarly, if:

B is the abbreviation for `*>BDIR>%1%`

and:

C is the abbreviation for `*>CDIR>%1%`

then:

(A B C) D E F

expands to:

`(*>SUBDIR>D *>BDIR>D *>CDIR>D) E F`

In this example, the list (A B C) is treated as a single token having 1 parameter, which is assigned the value D.

Function calls are also treated as single tokens, but they have the additional property that they are expanded as if they were a separate command line. Thus, the first token following a left bracket is considered to be in the command position, and no abbreviation parameters are taken from beyond the matching right bracket.



For example, if:

```
A = FOO %2%.TWO %1%.ONE
```

then:

```
[A B] C D
```

expands to:

```
[FOO .TWO B.ONE] C D
```

Note that C does not become the second parameter of A because C lies outside the function call brackets.

### SYNTAX SUPPRESSION

Following abbreviation expansion, the command processor checks to see if the first character on the command line is the syntax suppressor, the tilde (~). If so, processing of all the features described in the rest of this section is suppressed. Then the tilde is removed, and the remaining command line is executed as is.

For example, the command:

```
COMO LOG.DATE [DATE -MONTH]
```

opens a command output file in the current directory called LOG.MAY (during the month of May).

In contrast, the command:

```
~ABBREV -ADD_COMMAND LOG LOG.[DATE -MONTH]
```

will add the abbreviation LOG to your abbreviations file, which stands for LOG.[DATE -MONTH]. The function is inserted into the abbreviations file without being evaluated. Each time you subsequently use the abbreviation log, the abbreviation will be expanded and the expansion will be evaluated. For example, the command:

```
COMO LOG
```

will expand to:

```
COMO LOG.[DATE -MONTH]
```

and will create a command output file in your directory called (during the month of February):

```
LOG.FEBRUARY
```



MULTIPLE COMMAND PROCESSING

The command processor next scans the command line for the command separator character, the semi-colon (;). This character delimits multiple commands on the same command line. The recognition of the command separator is disabled if the syntax suppressor was used, or if the command is ABBREV or AB. The latter exception is provided so that it is easy to define new abbreviations whose value contains the command separator character. For example:

```
ABBREV -AC ZOT CLOSE ALL; DELETE @@ -NO_VERIFY
```

is a single command that defines an abbreviation, "ZOT", whose value is "CLOSE ALL; DELETE @@ -NO\_VERIFY".

If recognition of the command separator is not disabled, the features described below are executed separately for each subcommand on the command line. For example, in the following command line:

```
command1 [function1 args]; command2 [function2 args]
```

the order of execution is: evaluate function1; execute command1; evaluate function2; execute command2.

Note

Because of the way the COMINPUT command operates, it should not be used in command lines containing multiple commands.

VARIABLE AND FUNCTION EVALUATION

Once the current command has been identified, variable references are evaluated. Each reference of the form "%variable\_name%" is replaced by the value of "variable\_name".

Next, command function references of the form "[function arguments]" are evaluated and replaced by their values. Evaluation proceeds from the inside out.

For example, following command line:

```
command [function arguments]
```

will be replaced by:

```
command value
```

where value is the value returned by function. (See the CPL User's Guide or PRIMOS Commands Reference Guide for details on functions.)



The fact that variables and functions are evaluated after the command separator has been processed means that semicolons are not recognized as command separators when they form part of the value of a variable or function.

The fact that functions are evaluated after variables means that, if the value of a variable contains a function reference, the function will be evaluated. Conversely, if the value of a function contains a variable reference, the variable reference will not be evaluated, since variable evaluation has already occurred.

If any error occurs during variable or function evaluation, such as a reference to an undefined variable or function, the command processor prints an error message and does not process that subcommand.

### ITERATION

All iteration sets (lists in parentheses) in the command line are identified. Conceptually, the command processor can be thought of as producing a series of command lines, one for each iteration specified by the simple iteration sets. For example:

command (A B C) Y (D E F)

can be thought of as the series of commands:

command A Y D  
command B Y E  
command C Y F

In actual fact the command processor does not generate the command strings at this time; it deals with a kind of list structure at this level.

### TREEWALKING

Next, the command line is examined to see if it contains a treewalk pathname. This is a pathname whose directory part (the part before the final name) contains a wildcard. There may be at most one such pathname per command. (There may be more than one on the command line, as long as each iteration yields no more than one.)

The command processor opens the directory whose pathname appears before the directory wildcard in the treewalk pathname. It visits the subdirectories of the tree, as explained in the discussion on treewalking in the previous chapter, and substitutes the pathname of each directory visited for the part of the treewalk pathname to the left of and including the directory wildcard.



For example:

```
command A>B>@>@>@.LIST
```

might be thought of as executing as if it were the series of commands:

```
command A>B>C>@.LIST
command A>B>C>X>@.LIST
command A>B>C>Y>@.LIST
command A>B>D>@.LIST
```

and so forth.

Commands of this series are conceptually passed on to the wildcarding step, below.

### WILDCARDING

Each command is scanned to see if it contains a pathname whose last component (entryname) is a wildcard. There may be at most one of these per command received at this step.

At the same time, the command is scanned for wildcard arguments (such as -AFTER) which specify selection criteria that are applied in addition to the wildcard name. These wildcard arguments are used only at this step in processing. Therefore, they are removed from the commands which are actually executed.

The command processor opens the directory given by the treename, and selects those entries in the directory that match the wildcard and the other selection criteria. If verify mode is enabled, the command processor will ask the user to approve or disapprove each match.

The wildcard part of the pathname is then replaced with the actual name matched, and the command is passed on to the next step.

For example:

```
command @.LIST -AFTER 12-1 -FILE
```

might execute as if it were the series:

```
command A.LIST
command B.LIST
command C.LIST
```

.

.

.



NAME GENERATION

The final step is name generation. The command processor searches for any pathname in the command that contains the character = in the objectname. Any number of such pathnames is permitted. Each generation pattern, as a name containing an = is known, is replaced by the name it generates. The source name is usually the first object argument to the command, although individual commands may differ.

For example:

```
command ABC.LIST ==.+OLD
```

would execute as:

```
command ABC.LIST ABC.LIST.OLD
```

EXECUTION

The command that emerges from the name generation step is then actually executed. After execution, whether the command produced a positive severity code (ER! prompt) or not, the next wildcard match, then the next treewalk step, and finally the next simple iteration step, is taken.

AN EXAMPLE

Let us follow a sample command line, step by step, through processing. This will show the order in which command line processing occurs. (Remember, the stack of waiting command lines shown in this example would not be created by PRIMOS. The commands are shown in this form for easier reading.)

The sample directory used in this example is shown in Figure 19-1.

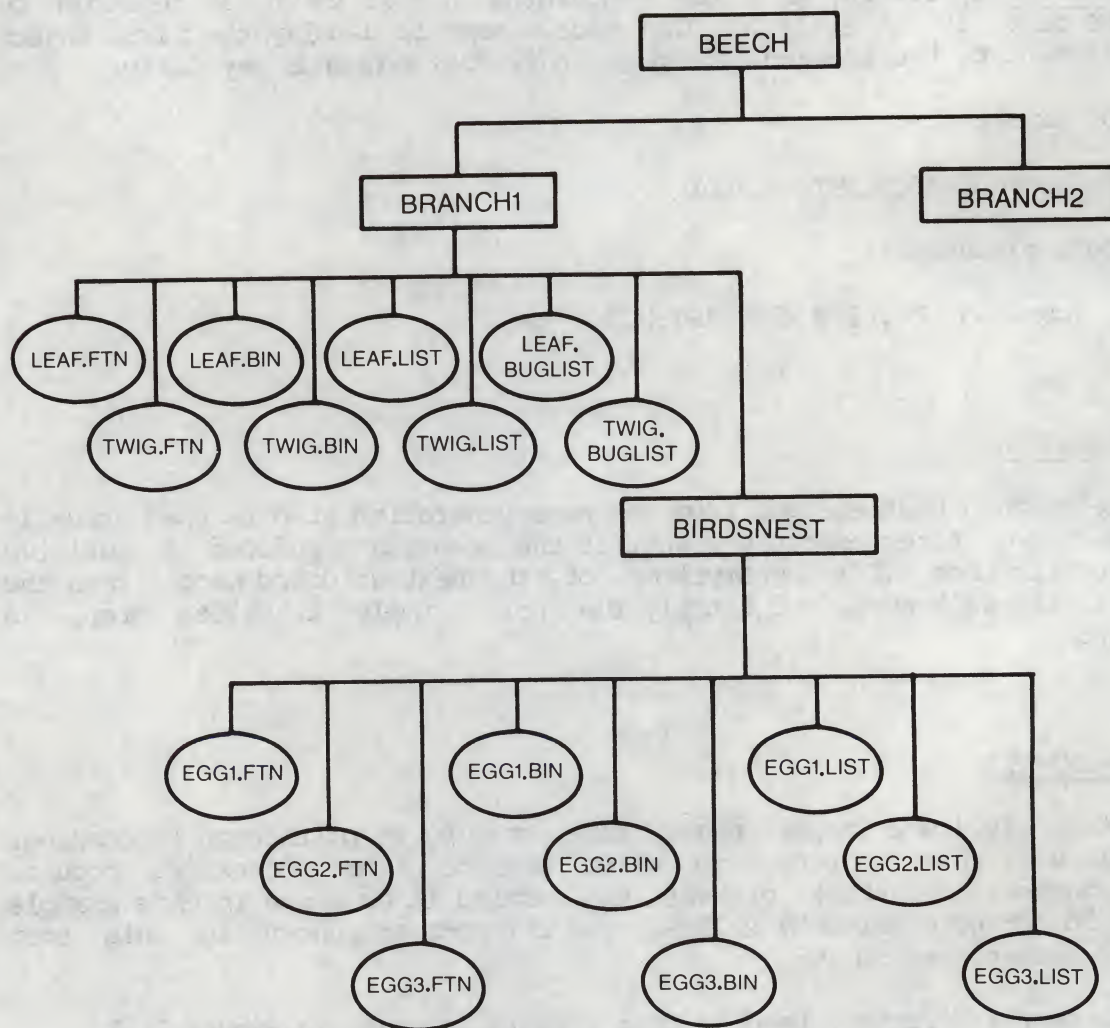
Processing Steps

1. The user types the command line: CHECK %.PROJ1%
2. Abbreviation Expansion: The command processor expands the user's abbreviation CHECK, producing the command line:

```
SPOOL %.PROJ1%>(@.LIST @.BUGLIST); DELETE %.PROJ1%>(@.BIN @.LIST)
```

3. Syntax Suppression: The command line does not begin with a tilde (^). No syntax suppression is done. Processing continues.





Sample Directory Tree  
Figure 19-1



4. Multiple Commands: The presence of a semicolon breaks the command line into two commands. You now have:

```

▶ CURRENT:   SPOOL %.PROJ1>(@.LIST @.BUGLIST)

▶ WAITING:   DELETE %.PROJ1>(@.BIN @.LIST)

```

5. Variable Evaluation: The global variable %.PROJ1% is removed from the current command and replaced with its value, BEECH>@@, from the global variable file previously activated. You now have:

```

▶ CURRENT:   SPOOL BEECH>@@>(@.LIST @.BUGLIST)

▶ WAITING:   DELETE %.PROJ1>(@.BIN @.LIST)

```

6. Command Function Evaluation: The command line contains no square brackets ( [ ] ). Therefore, it contains no function calls. No change to the command occurs here.

7. Iteration Lists: Next, the iteration lists are evaluated. The presence of an iteration list containing two items -- (@.LIST @.BUGLIST) -- creates two command lines to replace the current command line. The first of the two remains current. The second goes onto the top of the waiting list. Thus, you have:

```

▶ CURRENT:   SPOOL BEECH>@@>@.LIST

▶ WAITING:   SPOOL BEECH>@@>@.BUGLIST
             DELETE %.PROJ1>(@.BIN @.LIST)

```

8. Treewalking: Next, treewalking evaluation matches @@ against all directories in BEECH's subtree. Each match found creates a new command line. The first becomes current. The rest go onto the top of the waiting list. You now have:

```

▶ CURRENT:   SPOOL BEECH>BRANCH1>@.LIST

▶ WAITING:   SPOOL BEECH>BRANCH1>BIRDSNEST>@.LIST
             SPOOL BEECH>BRANCH2>@.LIST
             SPOOL BEECH>@@>@.BUGLIST
             DELETE %.PROJ1>(@.BIN @.LIST)

```



9. Wildcard Matching: Matching of wildcards is done for the current command. The first match found replaces the wildcard in the current command line. Subsequent matches generate new commands for the waiting list.

You now have:

```

► CURRENT:  SPOOL BEECH>BRANCH1>LEAF.LIST

► WAITING:  SPOOL BEECH>BRANCH1>TWIG.LIST
            SPOOL BEECH>BRANCH1>BIRDSNEST>@.LIST
            SPOOL BEECH>BRANCH2>@.LIST
            SPOOL BEECH>@>@.BUGLIST
            DELETE %.PROJ1%>(@.BIN @.LIST)

```

10. Name Generation: The current command is checked for name generation patterns. There are none, so no changes occur at this step.

11. Execution: The current command:

```
SPOOL BEECH>BRANCH1>LEAF.LIST
```

is executed.

12. The first command on the waiting list becomes "current". You now have:

```

► CURRENT:  SPOOL BEECH>BRANCH1>TWIG.LIST

► WAITING:  SPOOL BEECH>BRANCH1>BIRDSNEST>@.LIST
            SPOOL BEECH>BRANCH2>@.LIST
            SPOOL BEECH>@>@.BUGLIST
            DELETE %.PROJ1%>(@.BIN @.LIST)

```

13. The current command is checked for name generation patterns, then executed. The next command then moves up, so that you have:

```

► CURRENT:  SPOOL BEECH>BRANCH1>BIRDSNEST>@.LIST

► WAITING:  SPOOL BEECH>BRANCH2>@.LIST
            SPOOL BEECH>@>@.BUGLIST
            DELETE %.PROJ1%>(@.BIN @.LIST)

```



14. Wildcard matching on the current command produces:

```

▶ CURRENT:  SPOOL BEECH>BRANCH1>BIRDSNEST>EGG1.LIST

▶ WAITING:  SPOOL BEECH>BRANCH1>BIRDSNEST>EGG2.LIST
            SPOOL BEECH>BRANCH1>BIRDSNEST>EGG3.LIST
            SPOOL BEECH>BRANCH2>@.LIST
            SPOOL BEECH>@@>@.BUGLIST
            DELETE %.PROJ1%>(@.BIN @.LIST)

```

15. The current command, and then the next two, are checked for name generation, then executed. This gives you boxes 15 through 17:

```

▶ CURRENT:  SPOOL BEECH>BRANCH1>BIRDSNEST>EGG2.LIST

▶ WAITING:  SPOOL BEECH>BRANCH1>BIRDSNEST>EGG3.LIST
            SPOOL BEECH>BRANCH2>@.LIST
            SPOOL BEECH>@@>@.BUGLIST
            DELETE %.PROJ1%>(@.BIN @.LIST)

```

16. Next:

```

▶ CURRENT:  SPOOL BEECH>BRANCH1>BIRDSNEST>EGG3.LIST

▶ WAITING:  SPOOL BEECH>BRANCH2>@.LIST
            SPOOL BEECH>@@>@.BUGLIST
            DELETE %.PROJ1%>(@.BIN @.LIST)

```

17. Next:

```

▶ CURRENT:  SPOOL BEECH>BRANCH2>@.LIST

▶ WAITING:  SPOOL BEECH>@@>@.BUGLIST
            DELETE %.PROJ1%>(@.BIN @.LIST)

```



18. The current command requires wildcard matching. However, there are no wildcard matches for the directory BRANCH2. The current command line never executes, and the next command becomes current. You now have:

```

▶ CURRENT:  SPOOL BEECH>@>@.BUGLIST

▶ WAITING:  DELETE %.PROJ1%>(@.BIN @.LIST)

```

19. Treewalking gives you:

```

▶ CURRENT:  SPOOL BEECH>BRANCH1>@.BUGLIST

▶ WAITING:  SPOOL BEECH>BRANCH1>BIRDSNEST>@.BUGLIST
             SPOOL BEECH>BRANCH2>@.BUGLIST
             DELETE %.PROJ1%>(@.BIN @.LIST)

```

20. Wildcard expansion gives you:

```

▶ CURRENT:  SPOOL BEECH>BRANCH1>LEAF.BUGLIST

▶ WAITING:  SPOOL BEECH>BRANCH1>TWIG.BUGLIST
             SPOOL BEECH>BRANCH1>BIRDSNEST>@.BUGLIST
             SPOOL BEECH>BRANCH2>@.BUGLIST
             DELETE %.PROJ1%>(@.BIN @.LIST)

```

21. The two top commands are each checked for name generation, then executed, giving you items 21 and 22:

```

▶ CURRENT:  SPOOL BEECH>BRANCH1>TWIG.BUGLIST

▶ WAITING:  SPOOL BEECH>BRANCH1>BIRDSNEST>@.BUGLIST
             SPOOL BEECH>BRANCH2>@.BUGLIST
             DELETE %.PROJ1%>(@.BIN @.LIST)

```



22. Next:

► CURRENT: SPOOL BEECH>BRANCH1>BIRDSNEST>@.BUGLIST

► WAITING: SPOOL BEECH>BRANCH2>@.BUGLIST  
DELETE %.PROJ1%>(@.BIN @.LIST)

23. Wildcard matching produces two more null matches. The top two commands do not execute, and the next command becomes current:

► CURRENT: DELETE %.PROJ1%>(@.BIN @.LIST)

The current command now goes through the following stages:

24. Variable Evaluation:

► CURRENT: DELETE BEECH>@@>(@.BIN @.LIST)

25. Function Evaluation: (There are no function calls.)

26. Iteration Checking:

► CURRENT: DELETE BEECH>@@>@.BIN

► WAITING: DELETE BEECH>@@>@.LIST

27. Treewalking:

► CURRENT: DELETE BEECH>BRANCH1>@.BIN

► WAITING: DELETE BEECH>BRANCH1>BIRDSNEST>@.BIN  
DELETE BEECH>BRANCH2>@.BIN  
DELETE BEECH>@@>@.LIST



28. Wildcard Matching:

► CURRENT: DELETE BEECH>BRANCH1>LEAF.BIN

► WAITING: DELETE BEECH>BRANCH1>TWIG.BIN  
DELETE BEECH>BRANCH1>BIRDSNEST>@.BIN  
DELETE BEECH>BRANCH2>@.BIN  
DELETE BEECH>@>@.LIST

29. Name Generation Checking and Execution:

► CURRENT: DELETE BEECH>BRANCH1>TWIG.BIN

► WAITING: DELETE BEECH>BRANCH1>BIRDSNEST>@.BIN  
DELETE BEECH>BRANCH2>@.BIN  
DELETE BEECH>@>@.LIST

30. Next:

► CURRENT: DELETE BEECH>BRANCH1>BIRDSNEST>@.BIN

► WAITING: DELETE BEECH>BRANCH2>@.BIN  
DELETE BEECH>@>@.LIST

31. More wildcard matching:

► CURRENT: DELETE BEECH>BRANCH1>BIRDSNEST>EGG1.BIN

► WAITING: DELETE BEECH>BRANCH1>BIRDSNEST>EGG2.BIN  
DELETE BEECH>BRANCH1>BIRDSNEST>EGG3.BIN  
DELETE BEECH>BRANCH2>@.BIN  
DELETE BEECH>@>@.LIST

32. The next group of executions:

► CURRENT: DELETE BEECH>BRANCH1>BIRDSNEST>EGG2.BIN

► WAITING: DELETE BEECH>BRANCH1>BIRDSNEST>EGG3.BIN  
DELETE BEECH>BRANCH2>@.BIN  
DELETE BEECH>@>@.LIST



33. Next:

```
► CURRENT:  DELETE BEECH>BRANCH1>BIRDSNEST>EGG3.BIN

► WAITING:  DELETE BEECH>BRANCH2>@.BIN
            DELETE BEECH>@@>@.LIST
```

34. Next:

```
► CURRENT:  DELETE BEECH>BRANCH2>@.BIN

► WAITING:  DELETE BEECH>@@>@.LIST
```

35. Nothing matches the wildcard, so the DELETE command does not execute. You now have:

```
► CURRENT:  DELETE BEECH>@@>@.LIST
```

36. The last treewalk:

```
► CURRENT:  DELETE BEECH>BRANCH1>@.LIST

► WAITING:  DELETE BEECH>BRANCH1>BIRDSNEST>@.LIST
            DELETE BEECH>BRANCH2>@.LIST
```

37. Wildcard Matching:

```
► CURRENT:  DELETE BEECH>BRANCH1>LEAF.LIST

► WAITING:  DELETE BEECH>BRANCH1>TWIG.LIST
            DELETE BEECH>BRANCH1>BIRDSNEST>@.LIST
            DELETE BEECH>BRANCH2>@.LIST
```



38. Execution:

► CURRENT: DELETE BEECH>BRANCH1>TWIG.LIST

► WAITING: DELETE BEECH>BRANCH1>BIRDSNEST>@.LIST  
DELETE BEECH>BRANCH2>@.LIST

39. Wildcard Matching:

► CURRENT: DELETE BEECH>BRANCH1>BIRDSNEST>@.LIST

► WAITING: DELETE BEECH>BRANCH2>@.LIST

40. The next group of executions:

► CURRENT: DELETE BEECH>BRANCH1>BIRDSNEST>EGG1.LIST

► WAITING: DELETE BEECH>BRANCH1>BIRDSNEST>EGG2.LIST  
DELETE BEECH>BRANCH1>BIRDSNEST>EGG3.LIST  
DELETE BEECH>BRANCH2>@.LIST

## 41. Next:

► CURRENT: DELETE BEECH>BRANCH1>BIRDSNEST>EGG2.LIST

► WAITING: DELETE BEECH>BRANCH1>BIRDSNEST>EGG3.LIST  
DELETE BEECH>BRANCH2>@.LIST

## 42. Next:

► CURRENT: DELETE BEECH>BRANCH1>BIRDSNEST>EGG3.LIST

► WAITING: DELETE BEECH>BRANCH2>@.LIST

43. The last wildcard matching:

► CURRENT: DELETE BEECH>BRANCH2>@.LIST



44. Nothing matches the wildcard, so the DELETE command does not execute.

45. The End

## Terminal Display

When the command in the example executes, the user sees the response at the terminal shown in Figure 19-2.



```

OK, CHECK %.PROJ%
[SPOOL rev 19.0]
PRT001 spooled, records: 1, name: BEECH>BRANCH1>LEAF.LIST
[SPOOL rev 19.0]
PRT002 spooled, records: 1, name: BEECH>BRANCH1>TWIG.LIST
[SPOOL rev 19.0]
PRT003 spooled, records: 1, name: BEECH>BRANCH1>BIRDSNEST>EGG1.LIST
[SPOOL rev 19.0]
PRT004 spooled, records: 1, name: BEECH>BRANCH1>BIRDSNEST>EGG2.LIST
[SPOOL rev 19.0]
PRT005 spooled, records: 1, name: BEECH>BRANCH1>BIRDSNEST>EGG3.LIST
[SPOOL rev 19.0]
PRT006 spooled, records: 1, name: BEECH>BRANCH1>LEAF.BUGLIST
[SPOOL rev 19.0]
PRT007 spooled, records: 1, name: BEECH>BRANCH1>TWIG.BUGLIST
(std$cp) Verify wildcard selections for "BEECH>BRANCH1>@.BIN":
"LEAF.BIN"? YES
"TWIG.BIN"? YES
(std$cp) Verify wildcard selections for "BEECH>BRANCH1>BIRDSNEST>@.BIN":
"EGG1.BIN"? YES
"EGG2.BIN"? YES
"EGG3.BIN"? YES
(std$cp) Verify wildcard selections for "BEECH>BRANCH1>@.LIST":
"LEAF.LIST"? YES
"TWIG.LIST"? YES
(std$cp) Verify wildcard selections for "BEECH>BRANCH1>BIRDSNEST>@.LIST":
"EGG1.LIST"? YES
"EGG2.LIST"? YES
"EGG3.LIST"? YES
OK,

```

Sample Terminal Display  
Figure 19-2



# APPENDIXES



ALBERT DEARS



- Give the DMSTK command. This will print a stack dump, which traces the sequence of calls and returns by which the program reached its current state. The stack dump can be printed at the terminal or into a file, as the user prefers. If you are familiar with Prime machine architecture, you may find that the DMSTK command gives you enough information to solve your problem. (For details, see the PRIMOS Commands Reference Guide.) You may START a program again after dumping the stack.
- Give the DBG command to invoke the source-level debugger (if you have DBG). Then run the program again under DBG. If the DMSTK command did not provide enough information to solve the problem, this is probably the best course of action to take.
- Give the RLS command to release the errant program. You will remain at PRIMOS command level and can give any PRIMOS command you choose.

## Note

If the system default on-unit is invoked for a process running as a phantom or batch job, the condition mechanism prints the error message into the job's command output file and then logs the process out.

## ON-UNIT ACTIONS

On-units can:

- Terminate the program via a non-local GOTO, passing control back to the main program, so that it can call EXIT and return to PRIMOS level.
- Run diagnostic routines, then terminate the program (as above).
- Repair the problem which caused the error condition and have the program resume execution from the point of interrupt.
- Ignore the error condition and resume running the program.
- Transfer control to some predetermined spot in the program, possibly in a different procedure from the one which raised the error condition.
- "Continue to signal", passing control back to the condition mechanism and telling it to hunt for another on-unit.
- Print messages, then do any of the above.



THE CIPHER ALPHABET

The cipher alphabet is used to encipher and decipher messages.

When a message is enciphered, each letter of the message is replaced by a letter from the cipher alphabet. The cipher alphabet is a rearrangement of the alphabet. The key to the cipher is the cipher alphabet itself.

For example, if the cipher alphabet is A-Z, and the message is "HELLO", the enciphered message would be "KHOOR".

When a message is deciphered, each letter of the message is replaced by the letter from the cipher alphabet that corresponds to it. The key to the cipher is the cipher alphabet itself.

For example, if the cipher alphabet is A-Z, and the message is "KHOOR", the deciphered message would be "HELLO".

- 1. The cipher alphabet is used to encipher and decipher messages.
- 2. The cipher alphabet is a rearrangement of the alphabet.
- 3. The key to the cipher is the cipher alphabet itself.
- 4. For example, if the cipher alphabet is A-Z, and the message is "HELLO", the enciphered message would be "KHOOR".
- 5. When a message is deciphered, each letter of the message is replaced by the letter from the cipher alphabet that corresponds to it.
- 6. For example, if the cipher alphabet is A-Z, and the message is "KHOOR", the deciphered message would be "HELLO".



# A

## Glossary of Prime Concepts and Conventions

The following is a glossary of concepts and conventions basic to Prime computers, the PRIMOS operating system, and the file system. Cross references are underlined.

- access category

A file system object that contains only an access control list. It may be used to protect other file system objects.

- access control list

A list of users and the access privileges granted to each. Also called an ACL.

Access rights are:

P	protect
D	delete
A	add
L	list
U	use
R	read
W	write
ALL	all of the above
NONE	no access at all



When an ACL is associated with a file system object, it protects that object by allowing access only to the users listed within it, and allowing those users only their listed rights. See Chapter 16 for more information on access control lists.

- ACL

Same as access control list.

- argument

A variable that is related either to a command or to an option. For example, in the command line:

SPOOL pathname -DEFER time

pathname is an argument for the command SPOOL, and time is an argument for the option -DEFER.

- binary file

A translation of a source file generated by a language translator (FORTRAN, PL1, Pascal, F77, COBOL, PMA, RPG, VRPG). Such files are in the format required as input to the loaders. Also called "object file".

- byte

8 bits; 1 ASCII character.

- category name

The name of an access category. Category names follow the rules for objectnames.

- component

An elemental part of the name of a file system object. Components are separated by periods(.). For example, MYFILE and LIST are components of the filename MYFILE.LIST. See objectname for the list of legal characters. Objectnames with more than three components are not recommended, although up to sixteen components are allowed.



# 20

## Using the Condition Mechanism

### INTRODUCTION

PRIMOS has a condition mechanism which is activated when any executing process encounters certain unusual events. These events (or conditions) fall into three categories:

- Software-puzzling situations: illegal addresses, end of file encountered while reading data, and so on.
- Hardware and arithmetic exceptions: numbers too large or too small for the computer to handle, attempts to divide by zero, program too large for its allotted space, and so on.
- External occurrences: situations not directly controlled by the executing process, such as the use of the BREAK key from the user's terminal.

More than 30 PRIMOS-defined conditions exist. Some examples are:

<u>Condition</u>	<u>Definition</u>
ACCESS_VIOLATIONS	Process has attempted to read, write, or execute into a segment to which it has no access for that function.
ARITH\$	Arithmetic exception.



STACK_OVF\$	Process has overflowed its stack segment.
QUIT\$	User has hit break key on terminal.
ILLEGAL_INST\$	Process has tried to execute an illegal instruction.
ENDFILE (file)	End of file encountered while reading a PL/I file.

For a complete list of these conditions, see Appendix D. Full information on the condition mechanism appears in the Subroutines Reference Guide.

#### USING THE CONDITION MECHANISM

The condition mechanism's goal is either to repair the problem and restart the program, or to terminate the program in an orderly manner. To achieve this goal, the condition mechanism activates diagnostic or remedial subroutines (or PL/I Subset G begin blocks) called on-units.

Users writing in FORTRAN IV, FORTRAN 77, PL/I Subset G, or PMA can define their own on-units within the procedures for which they are intended. However, all users are automatically protected by PRIMOS' system on-units. When an error condition occurs, the condition mechanism looks for on-units within the executing procedure. If it finds none, or if the procedure's on-units call for further help, the condition mechanism searches first through any calling procedures' on-units and then through the system's on-units, activating the first appropriate on-unit it finds.

#### THE SYSTEM DEFAULT ON-UNIT

Of all the system on-units, the system default on-unit is the one most likely to be encountered by the user. This on-unit prints the following message at the user's terminal, then returns the user to PRIMOS command level:

```
Error:  condition "condition" raised at "address"
        [extra information]
```

The user may then take any one of the following actions:

- Give the START command. The condition mechanism will try to resume running the program from the point at which the condition was raised.



- condition mechanism

A PRIMOS facility which responds to conditions that would normally cause program termination. Rather than terminating the program immediately, the condition mechanism activates an on-unit to take some diagnostic or remedial action. A list of conditions handled by PRIMOS' condition mechanism is given in the Subroutines Reference Guide.

- CPU

Central Processing Unit (the Prime computer proper as distinct from peripheral devices or main memory).

- current directory

The directory to which you are currently attached.

- directory

A file system object that contains a list of objectnames, along with information on their characteristics and location. MFDs, UFDs, and subdirectories (sub-UFDs) are all directories. (To be distinguished from segment directory.)

- directory name

The name of a directory. Directory names follow rules for objectnames.

- device

A mechanical unit, such as a card reader, tape reader, or keyboard.

- entryname

Same as objectname.

- external command

A PRIMOS command existing as a runfile in the command directory (CMDNCO). It is invoked by name, and executes in user address space. No system-wide abbreviations exist for external commands. Users may define abbreviations for external commands by using the ABBREV command.



- file

An organized collection of information stored on a disk (or another peripheral storage medium, such as tape). Each file has an identifying label called a filename. Filenames follow the rules for objectnames.

- file system object

An organized collection of data stored on a disk (or a peripheral storage medium such as tape). Each object has an identifying label called an objectname. File system objects include files, directories, segment directories, and access categories.

- filename

The name of a file. Filenames follow the rules for objectnames. Directory names and a filename may be combined into a pathname. Most commands accept a pathname wherever a filename is required.

- file-unit

A number between 0 and 127 ('177, or octal 177) assigned as a pseudonym to each open file by PRIMOS. This number may be given in place of a filename in certain commands, such as CLOSE. PRIMOS-level internal commands require octal values. Each user is guaranteed at least 16 file units at a time. The maximum number of units that a user may have open simultaneously varies per installation; the default is 128. PRIMOS always reserves units 0 and 127 for its own use.

- file protection keys

See keys, file protection.

- identifier

The generic term for "user" listed in access control lists. Identifiers may be of three types: a user-id (for example, "SMITH"), a group name (for example, ".CLUB"), or the special identifier "\$REST" (that is, "everybody else").

- identity

The addressing mode plus its associated repertoire of computer instructions. Programs compiled in 32R or 64R mode execute in the R-identity; programs compiled in 64V mode execute in the V-identity. Programs compiled in 32I mode execute in the I-identity. R-identity, V-identity and I-identity are also then called R-mode, V-mode, and I-mode.



- initial attach point

Same as origin directory.

- internal command

A command that executes in PRIMOS address space. Does not overwrite the user memory image. PRIMOS-defined abbreviations exist for internal commands.

- keys, file protection

Specify file protection, as in the PROTECT command:

NIL	No access
R	Read
W	Write
RW	Read/Write
D	Delete
RD	Read and delete
WD	Write and delete
RWD	Read, write, and delete (all access)

- Ldev

Logical disk device number as printed by the command STATUS DISKS. (See ldisk.)

- ldisk

A parameter to be replaced by the logical unit number (octal) of a disk volume. It is determined when the disk is brought up by a STARTUP or ADDISK command. Printed as LDEV by STATUS DISKS.

- logical disk

A disk volume (or partition) that has been assigned a logical disk number either by the operator or during system startup.

- MFD

The Master File Directory. A special directory that contains the names of the UFDs on a particular disk or partition. There is one MFD for each logical disk.



- mode

An addressing scheme. The mode used determines the construction of the computer instructions by a compiler or assembler. (See identity.)

- nodename

Same as systemname.

- number representations

xxxxx	Decimal
'xxxxx	Octal
\$xxxxx	Hexadecimal

- object

Same as file system object.

- object file

Same as binary file.

- objectname

A sequence of 32 or fewer characters which names a file system object. Within any directory, each objectname is unique.

Objectnames may contain only the following characters:

A-Z, 0-9, \_ # \$ - . \* & /

The first character of an objectname must not be numeric. On some devices underscore (\_) prints as backarrow (<-). An objectname may sometimes be called an "entryname".



- objectname conventions

Suffixes indicate various types of files. (A period separates the suffix from the base name of the file.)

Objectname suffixes that are recognized by Prime Software include:

<u>Objectname with Suffix</u>	<u>Meaning</u>
filename.compiler-name	Source file (see list in Table 5-1)
filename.LIST	Listing file
filename.BIN	Binary (object) file
segdirectoryname.SEG	V-mode runfile (executable)
filename.SAVE	R-mode runfile (executable)
filename.CPL	CPL file
categoryname.ACAT	Access category
filename.CDML	COBOL Data Manipulation Language (CDML) preprocessor input file
filename.FDML	FORTTRAN Data Manipulation Language (FDML) preprocessor input file
filename.DPTCFG	DPTCFG input file
filename.EFASL	EMACS fastload file
filename.EM	EMACS extension file
filename.FBIN	FORMS object file
filename.FORM	FORMS file
filename.ERROR	Error output file (used by VPTCFG, CDML, FDML)

Other suffixes, which are not recognized by Prime Software but are used by Prime and are recommended to order and clarify your work, include:

<u>Objectname with Suffix</u>	<u>Meaning</u>
filename.ABBREV	Abbreviation file
filename.COMI	Command input file or phantom file
filename.COMO	Command output file
filename.CONFIG	Configuration file
filename.GVAR	Global-variable file
filename.MAP	Map file (created by LOAD or SEG)
filename.T	Temporary file
ffilename.RUNI	Runoff source (input) file
filename.RUNO	Runoff output file
filename.ERR	Error message text file
filename.HELP	Help text file
filename.INS.language	Insert and include files
filename.LOG	System activity log (used by FTS, VISTA, OAS, POWERPLUS)



A previous convention, which used prefixes to designate some file types, is still supported for compatability. Filenames under this convention are as follows:

<u>Objectname</u>	<u>Meaning</u>
B_filename	Binary (object) file
C_filename	Command input file
L_filename	Listing file
M_filename	Load map file
O_filename	Command output file
PH_filename	Phantom command file
filename	Source file or text file
*filename	SAVED (executable) R-mode runfile
#segdirectoryname	Stored (executable) V-mode runfile

- on-unit

A begin block (in PL/I Subset G) or subroutine (in FORTRAN or PL/I Subset G) which is activated by the condition mechanism to handle error conditions. PRIMOS has on-units for all conditions it recognizes. Users may also define on-units within any procedure they write. User-written on-units take precedence over system ones.

- open

Active state of a file-unit. A command or program opens a file-unit in order to read or write it.

- option

A PRIMOS term, preceded by a hyphen and related to a command. It signals an activity. For example, in the command line:

SPPOOL -LIST

-LIST is the option that tells the spooler to list at your terminal the files waiting to be printed.

- origin directory

The directory to which you are attached when you log in. Also called your "initial attach point".



- output stream

Output from the computer that would usually be printed at a terminal during command execution, but which is also written to a file if COMOUTPUT command was given.

- packname

See volume-name.

- page

A block of 1024 16-bit words within a segment (512 words on Prime 300).

- partition

A portion (or all) of a multihead disk pack. Each partition is treated by PRIMOS as a separate logical device. Partitions are an integral number of heads in size, offset an even number of heads from the first head. A volume occupies a partition, and a "partition of a disk" and a "volume of files" are actually the same thing.

- pathname

A multi-part name which uniquely specifies a particular file system object within a file system tree. A pathname (also called treename) gives a path from the disk volume, through directory and subdirectories, to a particular object. See the discussion on Pathnames in Chapter 2.

- Pdev

Physical disk unit number as printed by STATUS DISKS. (See pdisk.)

- pdisk

A parameter to be replaced by a physical disk unit number. Needed only for operator commands.

- phantom user

A process running independently of a terminal, under the control of a CPL program or a command file.



- procedure

In FORTRAN, a subroutine or function. In PL/I Subset G, any subroutine, function, or program. (In PL/I Subset G, procedures may contain other procedures.) In COBOL, the term usually refers to one or more related paragraphs or sections within the Procedure Division. Procedures direct the computer to perform a particular operation or a series of operations.

- process

A particular program running in a particular address space.

- quota

The maximum number of records (1 record = 1024 user data words) that the contents of a directory can occupy on a disk. A quota of 0 means that no record limit exists on the directory.

- R-mode

The addressing mode in which a single segment program is generated. A single segment program in R-mode cannot exceed 128K bytes and cannot use virtual addressing. R-mode is used mainly to create external commands.

- reserved characters

The following characters are reserved by PRIMOS for special uses. They may not be used in objectnames.

= ; , ( ) ` [ ] ! { } ^ " ? : ~ | < > @ + ' % \  
space delete/rubout

- runfile

Executable version of a program, consisting of the loaded binary file, subroutines and library entries used by the program, COMMON areas, initial settings, etc. (Created using LOAD or SEG.)

- SEG

Prime's segmented loading utility.



- segment

A 65,536-word block of address space.

- segment directory

A special form of directory used in direct-access file operations and V-mode run images. Not to be confused with directory, which refers only to MFDs, UFDs, and sub-UFDs.

- segno

Segment number; the "name" that identifies a segment.

- source file

A file containing programming language statements in the format required by the appropriate compiler or assembler.

- subdirectory

A directory that is in a UFD or another subdirectory.

- sub-UFD

Same as subdirectory.

- suffixes, objectname

See objectname conventions.

- systemname

The name of a system on a network; assigned when a local PRIMOS system is built or configured.

- treename

Same as pathname.

- UFD

A User File Directory, one of the directories listed in the MFD of a volume.



- unit

See file-unit.

- V-mode

The addressing modes in which a multi-segmented program is normally generated. Programs in V-mode can be as large as 32MB and can take full advantage of the virtual address space.

- volume

A self-sufficient unit of disk storage, including an MFD, a disk record availability table, and associated files and directories. A volume may occupy a complete disk pack or be a partition within a multi-head disk pack.

- volume-name

A sequence of 6 or fewer characters labeling a volume. The name is assigned during formatting (by MAKE). The STATUS DISKS command uses this name in its DISK column to identify the disk.

- word

As a unit of address space, two bytes or 16 bits.



# B

## System Defaults and Constants

Prime systems have the following defaults and constants.

### TERMINAL DEFAULTS

Full duplex

X-ON/X-OFF disabled

Buffered protocol (using carrier detect) disabled

Input error checking disabled

### PRIMOS KEYBOARD STANDARDS

<u>Character</u>	<u>Octal Value</u>	<u>Interpretation</u>
CTRL-P	'220	.BREAK. (Interrupt)
.CR.	'215	Newline (.NL.). Newline is converted to '212 (line feed) by standard software.



PRIMOS KEYBOARD DEFAULTS

<u>Character</u>	<u>Interpretation</u>
"	Character erase
?	Line kill

Note

Users may change their erase and kill characters with the TERM command.

PRIMOS COMMAND LINE STANDARDS

<u>Character</u>	<u>Interpretation</u>
+	1. Wild (1 character); 2. Add literal (name generation)
@	Wild (1 or more characters)
^	1. Wild (negation); 2. Exclusion (name generation)
=	Name generation
;	Command separator
~	Suppresses command processor syntax

EDITOR DEFAULTS

INPUT (TTY)

LINESZ 144

MODE NCKPAR

MODE NCOLUMN

MODE NCOUNT

MODE NNUMBER



MODE NPROMPT

MODE PRALL

VERIFY

### ED Editor Symbols

<u>Name</u>	<u>Character</u>	<u>Interpretation</u>
BLANKS	#	Match <u>n</u> spaces (with FIND, NFIND)
COUNTER	@	MODE COUNT's counter symbol
CPROMPT	\$	MODE PROMPT's edit prompt
DPROMPT		MODE PROMPT's input prompt
ERASE	"	Character erase
ESCAPE	^	Logical escape (see Notes below)
KILL	?	Line kill
SEMICO	;	End of line or command
TAB	\	Logical tab
WILD	!	Match any character (with FIND, NFIND)

### Notes

Users may change any of these special characters from within the ED editor with the SYMBOL command.

Non-printing characters may be entered into text with the ED editor by using the logical escape character (^ = default escape character) and the octal value. The non-printing character is interpreted by output devices according to their hardware. For example, typing ^207 (where ^ = escape) will enter one character into the text.



SEGMENTED LOADER (SEG)

<u>Item</u>	<u>Value</u>
Loading address	Current TOP+1 in current procedure segment
Stack size	'6000 words
Library	PFINLB and IFINLB libraries

VIRTUAL LOADER (LOAD)

<u>Item</u>	<u>Value</u>
Memory Location	'122770 - '144000
Loading address	Current *PBRK value
Library	FINLIB FORTRAN library
MODE	D32R
Sector Zero Base Area	Base start at location '200 Base range '600 words
COMMON	Top = '077777

EXECUTION

<u>Item</u>	<u>Value</u>
A-register value	0
B-register value	0
X-register value	0
Program start address	'1000
Bits 4-6 of Keys	000 16K, sector-address 001 32K, sector-address 010 64K, relative-address 011 32K, relative-address 110 64K, segmented-address



PROTECTION

File System Objects in ACL-protected Directories

Newly created files are opened with RW rights; once stored, they assume default protection (same protection existing on the directory where they reside). Other objects are created with default protection.

Files in Password-protected Directories

New files are created with the following protection:

owner: all access rights (RWD)  
non-owner: no access rights (NIL)







# C

## ASCII Character Set

The standard character set used by Prime is the ANSI, ASCII 7-bit set, shown in Tables C-1 and C-2. This character set conforms to ANSI X3.4-1968. (1963 variances are noted.)

### PRIME USAGE

Prime hardware and software uses standard ASCII for communications with devices. The following points are particularly important in Prime usage.

- The Prime internal standard for the parity bit is one; that is, the high order (parity) bit is always set.
- Output Parity is normally transmitted as a one (mark) since this is how characters are stored internally in Prime systems. If the device requires otherwise, software will compute transmitted parity. Some controllers (for example, MDLC) may have hardware to assist in parity generations.
- Input Parity is ignored by hardware and by standard software. Input drivers are responsible for making the parity bit suit the host software requirements. For normal terminal communication, the input driver forces the '200 bit on before passing the character to PRIMOS. Some controllers (for example, MDLC) may assist in parity error detection.



Table C-1  
ASCII Character Set (Non-Printing)

Octal Value	ASCII Char	Comments/Prime Usage	Control Char
200	NULL	Null character - filler	^@
201	SOH	Start of header (communications)	^A
202	STX	Start of text (communications)	^B
203	ETX	End of text (communications)	^C
204	EOT	End of transmission (communications)	^D
205	ENQ	End of I.D. (communications)	^E
206	ACK	Acknowledge affirmative (communications)	^F
207	BEL	Audible alarm (bell)	^G
210	BS	Back space one position (carriage control)	^H
211	HT	Physical horizontal tab	^I
212	LF	Line feed; ignored as terminal input	^J
213	VT	Physical vertical tab (carriage control)	^K
214	FF	Form feed (carriage control)	^L
215	CR	Carriage return (carriage control) (1)	^M
216	SO	Shift out (switch to alternate character set) (2)	^N
217	SI	Shift in (return to standard character set)	^O
220	DLE	Data link escape (3)	^P
221	DC1	Device control 1 (4)	^Q
222	DC2	Device control 2 (5)	^R
223	DC3	Device control 3 (6)	^S
224	DC4	Device control 4 (7)	^T
225	NAK	Negative acknowledgement (communications)	^U
226	SYN	Synchronization (communications)	^V
227	ETB	End of transmission block (communications)	^W
230	CAN	Cancel	^X
231	EM	End of Medium	^Y
232	SUB	Substitute	^Z
233	ESC	Escape	^[
234	FS	File separator	^[\
235	GS	Group separator	^]
236	RS	Record separator	^_
237	US	Unit separator	^-



Notes to Table C-1

- (1) Interpreted as .NL. (that is, line feed) at the application program level.
- (2) Examples of alternate character sets include red ribbon characters or graphics characters.
- (3) Has multiple functions, including:
  - From a terminal: aborts (quits) user programs and returns to PRIMOS level.
  - Within a file: specifies relative copy; next byte gives number of bytes to copy from corresponding position of previous line.
- (4) Has multiple functions, including:
  - From a terminal: XON; resume transmission.
  - Within a file: relative horizontal tab; next byte specifies number of spaces to insert.
- (5) Can have multiple functions, including:
  - Within a file: half line feed forward (carriage control).
- (6) Has multiple functions, including:
  - From a terminal: XOFF; suspends (freezes) transmission.
  - Within a file: relative vertical tab; next byte specifies number of lines to insert.
- (7) Can have multiple functions, including:
  - Within a file: half line feed reverse (carriage control).



Table C-2  
ASCII Character Set (Printing)

Octal Value	ASCII Character		OCTAL Value	ASCII Character		OCTAL Value	ASCII Character	
240	.SP.	(1)	300	@		340	`	(9)
241	!		301	A		341	a	
242	"	(2)	302	B		342	b	
243	#		303	C		343	c	
244	\$	(3)	304	D		344	d	
245	%		305	E		345	e	
246	&		306	F		346	f	
247	'	(4)	307	G		347	g	
250	(		310	H		350	h	
251	)		311	I		351	i	
252	*		312	J		352	j	
253	+		313	K		353	k	
254	,	(5)	314	L		354	l	
255	-		315	M		355	m	
256	.		316	N		356	n	
257	/		317	O		357	o	
260	0		320	P		360	p	
261	1		321	Q		361	q	
262	2		322	R		362	r	
263	3		323	S		363	s	
264	4		324	T		364	t	
265	5		325	U		365	u	
266	6		326	V		366	v	
267	7		327	W		367	w	
270	8		330	X		370	x	
271	9		331	Y		371	y	
272	:		332	Z		372	z	
273	;		333	[		373	{	
274	<		334	\		374		
275	=		335	]		375	}	
276	>		336	^	(7)	376	~	(10)
277	?	(6)	337	_	(8)	377	DEL	(11)



Notes to Table C-2

- (1) Space forward one position
- (2) Terminal usage: default erase character (erases previous character)
- (3) £ in British use
- (4) Apostrophe/single quote
- (5) Comma
- (6) Terminal usage: default kill character (kills line)
- (7) 1963 standard ↑ (up-arrow)
- (8) 1963 standard ← (back-arrow)
- (9) Grave
- (10) 1963 standard ESC
- (11) Rubout; ignored, unless the user has assigned it a particular action (for example, as his erase or kill character)



# 100-100000-1

100-100000-1

100-100000-1

100-100000-1

100-100000-1

100-100000-1

100-100000-1

100-100000-1

100-100000-1

100-100000-1

100-100000-1

100-100000-1

100-100000-1



# D

## Error Messages

### INTRODUCTION

Error messages are given in the following order:

- SEG Loader Error Messages
- LOAD Loader Error Messages
- Run-time Error Messages
- Batch Error Messages and Warnings
- Conditions Flagged by the Condition Mechanism

Errors are listed alphabetically within each group, according to the first word that is constant. Leading asterisks and variable names are ignored in alphabetizing.



SEG LOADER ERROR MESSAGES

- ATTEMPT TO REFERENCE UNDEFINED COMMON

This is a compilation problem. Try to recompile. If you still get the same message, see your administrator or analyst.

- BAD GROUP TYPE

This is a compilation problem. There are "groups" (that is, collections of instructions or data) in the binary code generated by the compiler. The compiler has encountered a group type that is not one of the recognized types. Try to recompile. If you still get the same message, see your administrator or analyst.

- BAD OBJECT FILE

User is attempting to load file which has faulty code. The file may not be an object file or it may be incorrectly compiled. Try to recompile. If you still get the same message, see your administrator or analyst.

- BASE AREA ZERO FULL

Extremely unlikely to occur. SEG uses two areas to store variables that it creates for unresolved references. One area is the sector base zero area. The other area is the area (or spaces) between your procedures. This message occurs when there is no more space in either area to store these variables. The more procedures you have in your program, the more space you have to store these variables. Break your large procedures into smaller procedures so that you will have more space. Then try to recompile. If you still get the same message, see your administrator or analyst.

Give the SZ command following this message to get the name and location of one procedure you should break down into smaller units. See the "sname xxxx/xxxx NEED SECTOR ZERO LINK" message below.

- BLOCK SIZE MISMATCH

This is a compilation problem. The message indicates that two internal size readings should match, but do not. Try to recompile. If you still get the same message, see your administrator or analyst.

- CAN'T LOAD IN SECTORED MODE

The Loader is attempting to load code in sectored mode that has not been compiled in sectored mode. This could arise if trying to load a



module compiled or assembled in 16S or 32S mode. It is unlikely that the average applications programmer will encounter this. Fatal error; the load must be aborted.

- CAN'T LOAD IN 64V OR 64R MODE

The Loader is attempting to load code in 64V mode which is not compiled in that mode. This would arise if:

1. A program was compiled in a mode other than 64V. In this case, you should recompile the program.
2. A PMA subroutine or procedure is written in code other than 64V and its mode is not specified. In this case (which the average applications programmer is unlikely to encounter), the PMA subroutine or procedure must be modified. This is a fatal error; the load must be aborted.

- COMMAND ERROR

An unrecognized command was entered, or the filenames or parameters following the command are incorrect. Usually not fatal.

- DEBUG GROUP ENCOUNTERED BEFORE A PROC DEF GROUP

This is a compilation problem. There are "groups" (that is, collections of instructions or data) in the binary code generated by the compiler. A "debug" group is used by DBG (Source Level Debugger) to examine programs. This type of group has been encountered before a "procedure definition" (PROC DEF) group. This sequence is not allowed. Try to recompile. If you still get the same message, see your administrator or analyst.

- \*\*EMPTY FILE\*\*

The file you have just tried to load is empty. It is possible that your compilation aborted.

- EXTERNAL MEMORY REFERENCE TO ILLEGAL SEGMENT

An attempt was made to load a 64R mode program, causing a reference to an illegal segment number. Recompile in 64V mode. Fatal error; the load must be aborted.

- ILLEGAL ADDRESSING MODE

This is a compilation problem. Try to recompile. If you still get the same message, see your administrator or analyst.



- ILLEGAL BLOCK TYPE

This is a compilation problem. Try to recompile. If you still get the same message, see your administrator or analyst.

- xxxx/xxxx: ILLEGAL INDEX OR INDIRECT ON AN ADDRESS CONSTANT

This is a compilation problem. Your program attempted to "index or go indirect" (that is, to use an index). The compiler selected a constant and tried to use it as an address, which is not allowed. This constant is located at location xxxx/xxxx, given in the error message. Try to recompile. If you still get the same message, see your administrator or analyst.

- sname: ILLEGAL REDEFINITION OF COMMON

An attempt is being made to redefine COMMON block sname to a longer length. Examine your program for consistent COMMON definitions. At the very least, the longest definition for a COMMON block should be first.

- sname: ILLEGAL REDEFINITION OF SHORT COMMON TO LONG COMMON

An attempt is being made to redefine COMMON block sname to a longer length. The original COMMON block sname was less than or equal to one segment in length. The new COMMON block sname is more than one segment long. Examine your program for consistent COMMON definitions. The longest definition for a COMMON block should be first.

- ILLEGAL SPLIT ADDRESS

Incorrect use of the Loader's SPLIT command. Segments may be split at '4000 boundaries only (i.e., '4000, '10000, '14000, etc.) Not fatal; resplit the segment.

- MEMORY REFERENCE TO COMMON IN ILLEGAL SEGMENT

An attempt was made to load a 64R mode program wherein COMMON would be allocated to an illegal segment number. Recompile in 64V mode. Fatal error; the load must be aborted.

- MISSING PROCEDURE END GROUP

This is a compilation problem. There are "groups" (that is, collections of instructions or data) in the binary code generated by the compiler. Each procedure must have a "procedure end group." The end group for the current procedure is missing. Try to recompile. If you still get the same message, see your administrator or analyst.



- xxxx/xxxx MULTIPLE INDIRECT

A subroutine or procedure loading in 64R mode requires a second level of indirection at location xxxx/xxxx. This message usually results when an attempt is made to load code compiled or assembled for 32R mode in 64R mode. It can also happen if code has accidentally been loaded into base areas as the result of a bad load command sequence. Break your program into smaller procedures and recompile.

- sname xxxx/xxxx NEED SECTOR ZERO LINK

A link is required at location xxxx/xxxx for desectoring the instruction. No base areas are within reach except sector zero. The last referenced symbol was sname. This message is only generated when the SZ command has been given. sname may be the name of a COMMON block, the name of the routine to which the link should be made, or the name of the module being loaded. Break your program (sname) into smaller procedures and recompile.

- NO FREE SEGMENTS TO ASSIGN

All SEG's segments have been allocated; no more are available at present. Fatal error; the load must be aborted. Use SYMBOL command to eliminate COMMON from assigned segments, thus reducing the number of assigned segments required. (User may need larger version of SEG and PRIMOS.) Try to rerun.

- NO ROOM IN SEGMENT

After SEG has placed a procedure into a segment, it tries to place the base area in the same segment as well. This message indicates that there is not enough room for both the procedure and the base area. The message normally appears because other procedures have also been placed in the same segment and little room is left. Use MAP to determine which segment SEG is trying to overload. Then move the procedure to the next segment and reload. If the procedure itself is close to one segment in length, break it into smaller procedures.

- NO ROOM IN SYMBOL TABLE

Unlikely to occur; no user solution. A new issue of SEG with a bigger symbol table is required. Check with analyst. As a temporary measure, user may try to reduce number of symbols used in program. Fatal error; the load must be aborted.



- OLD OBJECT FILE

SEG has encountered a subroutine or procedure created with a pre-Rev. 14 compiler or assembler (language processor). The subroutine or procedure must be recreated with a Rev. 15 or later compiler or assembler.

- OLD OBJECT FILE -- MIX FAILS

You have used the MIX command, and SEG has encountered a subroutine or procedure created with a pre-Rev. 14 compiler or assembler (language processor). The MIX command cannot be used with a pre-Rev. 14 compiler or assembler. The subroutine or procedure must be recreated with a Rev. 15 or later compiler or assembler.

- REFERENCE TO UNDEFINED SEGMENT

Almost always caused by improper use of the SYMBOL command to allocate initialized COMMON. Initialized COMMON cannot be located with the SYMBOL command; use R/SYMBOL or A/SYMBOL instead.

- SEGMENT WRAP AROUND TO ZERO

An attempt has been made to load a 64R mode program. The program has exceeded 128K bytes and is trying to be loaded over code previously loaded. Recompile in 64V mode. Fatal error; the load must be aborted.

- SYMBOL symbolname ALREADY EXISTS

You have tried to redefine the symbol symbolname (that is, a variable) in your dialog with SEG. symbolname has already been defined.

- SYMBOL symbolname NOT FOUND

You have tried to use the symbol symbolname (that is, a variable), which you have not defined in your dialog with SEG.

- WRONG FILE TYPE

SEG has encountered data that does not have the "group" structure of binary files that SEG expects. (A "group" is a collection of instructions or data.) You have probably tried to load a file that is not a binary file.



LOAD LOADER ERROR MESSAGES

- ALREADY EXISTS !

An attempt is being made to define a new symbol; however, the symbol name is already a defined symbol in the symbol table.

- BAD OBJECT FILE

The object text is not recognizable. This usually occurs when an attempt is made to load source code or when the object text was compiled or assembled for segmented loading.

- BASE SECTOR 0 FULL

All locations in the sector zero base area have been used. Use the AU command to generate base areas at regular intervals, or use the SETB or LOAD commands to place base areas specifically. Break your program into smaller procedures and recompile.

- CAN'T DEFER COMMON, OLD OBJECT TEXT

The DEFER COMMON command has been given, and a subroutine or procedure created with a pre-Rev. 14 compiler or assembler has been encountered. It is not possible to use DEFER COMMON in this case. The subroutine or procedure must be recreated with a Rev. 15 or later compiler or assembler.

- CAN'T - PLEASE SAVE

The EXECUTE command has been given for a run file which has required virtual loading. SAVE the runfile and give the EXECUTE command.

- CM\$

Command line error. Unrecognized command given. Not fatal.

- COMMON OUT OF REACH

Common above '100000 is out of reach of the current load mode (16S, 32S or 32R). Use the MODE command to set the load mode to 64R.



- COMMON TOO LARGE

Definition of this common block causes common to wrap around through zero. Moving the top of common with the COMMON command may help.

- sname xxxxxx ILLEGAL COMMON REDEFINITION

An attempt is being made at location xxxxxx to redefine COMMON block sname to a longer length. Examine your program for consistent COMMON definitions. The longest definition for a COMMON block should be first.

- xxxxxx MULTIPLE INDIRECT

A subroutine or procedure loading in 64R mode requires a second level of indirection at location xxxxxx. This message usually results when an attempt is made to load code compiled or assembled for 32R mode in 64R mode. It can also happen if code has accidentally been loaded into base areas as the result of a bad load command sequence. Break your program into smaller procedures and recompile.

- sname xxxxxx NEED SECTOR ZERO LINK

At location xxxxxx a link is required for desectoring the instruction. No base areas are within reach except sector zero. The last referenced symbol was sname. This message is only generated when the SZ command has been given. sname may be the name of a COMMON block, the name of the routine to which the link should be made, or the name of the routine being loaded. Break the program into smaller procedures and recompile.

- xxxxxx NO POST BASE AREA, OLD OBJECT TEXT

A post base area has been specified for module which was created with a pre-Rev. 14 compiler or assembler. No base area is created. Recreate the object text with a Rev. 15 or later compiler or assembler. This is not a fatal error.

- PROGRAM-COMMON OVERLAP

The module being loaded is attempting to load code into an area reserved for COMMON. Use the loader's COMMON command to move COMMON up higher.



- PROGRAM TOO LARGE

The program has loaded into the last location in memory and has wrapped around to load in Location 0. The program size must be decreased. Alternatively, compile in 64V mode and use SEG.

- REFERENCE TO UNDEFINED COMMON

An attempt is being made to link to a COMMON name which has not been defined. This usually happens to users creating their own translators.

- SECTORED LOAD MODE INVALID

A module compiled or assembled to load in R mode has been loaded in S mode. Use the MODE command to reset the load mode. Be sure that all modules are correctly written, since the default load mode is 32R.

- SYMBOL NOT FOUND

An attempt is being made to equate two symbols with the SYMBOL command and the old symbol does not exist.

- SYMBOL TABLE FULL

The symbol table has expanded down to location '4000. The last buffer cannot be assigned to the symbol table. Rebuild LOAD to load in higher memory locations, or reduce the number of symbols in the load.

- SYMBOL UNDEFINED

An attempt is being made to equate two symbols; however, the old symbol is an undefined symbol in the symbol table.

- 64R LOAD MODE INVALID

A module compiled or assembled to run in only 32K of memory is being loaded in 64R mode. Recompile, reassemble, or change the load mode with the loader's MODE command.



RUN-TIME ERROR MESSAGES

Descriptions of run-time error messages may be followed by labels enclosed either in parentheses or brackets. For example:

- Already exists. File System

Attempt made to create an object with the same name as one already existing. (CREA\$\$) (SRCH\$\$) [E\$EXST]

The label in parentheses is usually the name of a subroutine or group of subroutines; the label in brackets is the name of an error code. If a call to the subroutine is problematic, the error code is invoked. The error message is connected to the error code and is returned to (that is, printed at) the user's terminal.

- ACCESS VIOLATION 64V mode

Attempt to perform operations in segments to which user has no right.

- ACL too big. File System

Either ACL contains more than 32 entries, or the ACL exceeds space limitations. (AC\$SET, AC\$CHG) [E\$ACBG]

- \*\*\*\*AD R-mode function

Overflow or underflow in double-precision addition/subtraction (A\$66,S\$66).

- All file units in use. File System

User has requested use of a file unit when he already has the maximum allowable number of file units open, or the system has exhausted its pool of available units. (Search subroutines) [E\$FUIU]

- ALL REMOTE UNITS IN USE File System

Attempt made to assign a remote unit when none are available. (Network error) [E\$FUIU]

- \*\*\*\* ALOG/ALOG 10 - ARGUMENT <=0 V-mode function

Argument not greater than zero used in logarithm (ALOG, ALOG 10) function.



- filename ALREADY EXISTS Old file call

Attempt to create a file or UFD with the name of one already existing.  
[CZ]
- Already exists. File System

Attempt made to create an object with the same name as one already existing. (CREA\$\$, SRCH\$\$) [E\$EXST]
- \*\*\*\*AT R-mode function

Both arguments are zero in the ATAN2 function.
- \*\*\*\* ATAN2 - BOTH ARGUMENTS = 0 V-mode function

Both arguments are zero in the ATAN2 function.
- \*\*\*\* ATTDEV - BAD UNIT V-mode call

Incorrect logical device unit number in the ATTDEV subroutine call.
- BAD CALL TO SEARCH Old file call

Error in calling the SEARCH subroutine, e.g., incorrect parameter.  
[SA]
- Bad command format PRIMOS

User has issued an illegal command line. Command is ignored. [E\$CMND]
- BAD DAM FILE Old file call

The DAM file specified has been corrupted, either by the programmer or by a system problem. [SS]
- Bad DAM file. File System

The DAM file specified has been corrupted, either by the programmer or by a system problem. (PRWF\$\$, SRCH\$\$) [E\$BDAM]
- Bad dope vector. FORTRAN I/O

Your program may have overwritten itself. If not, this message may indicate a compiler or library error. [E\$BDV]



- Bad format.

FORTRAN I/O

Your program may have overwritten itself. If not, this message may indicate a compiler or library error. [E\$FER]

- Bad key in call.

PRIMOS

Incorrect key value specified in subroutine argument by a user's program. [E\$BKEY]

- Bad LUBTL entry.

FORTRAN I/O

Your program has possibly overwritten itself or part of the library. [E\$BLUE]

- BAD PARAMETER

Old file call

Incorrect parameter value in subroutine call. [SA]

- Bad parameter.

PRIMOS

Incorrect parameter value in subroutine call. [E\$BPAR]

- Bad password.

File System

Incorrect password specified in ATCH\$\$ subroutine. In pre-Rev. 19.0 systems, you return to PRIMOS level attached to no UFD. In Rev. 19.0 systems, you return to PRIMOS level attached to the home directory. The home directory is either the directory from which you activated the ATCH\$\$ subroutine, or another directory previously defined as home. (AT\$ subroutines) [E\$BPAS]

### Note

To protect UFD privacy the system does not allow the user to trap BAD PASSWORD errors.

- Bad remote password.

PRIMENET

An attempt was made to login to another system using an invalid password. [E\$BRPA]

- BAD RTNREC

PRIMOS

System error.



- Bad segment directory unit. File System

Error generated in accessing segment directory, i.e., PRIMOS file unit specified is not a segment directory. (SRCH\$\$) [E\$BSUN]

- Bad stack format. PRIMOS
- Bad stack format signalling.

Condition mechanism cannot perform requested action because the command processor stack has been damaged (system error). User is returned to PRIMOS command level. [E\$STKF, E\$STKS]

- BAD SVC PRIMOS

Bad supervisor call. In FORTRAN, usually caused by program writing over itself.

- Bad truncate of segment directory. File System

Error encountered in truncating segment directory. (SGDR\$\$) [E\$BITRAN]

- Bad unit number. File System

PRIMOS file unit number specified is invalid because it is outside legal range. (PRWF\$\$, RDEN\$\$, SRCH\$\$, SGDR\$\$). [E\$BUNT]

- Bad use of EXIT. PRIMOS

The condition mechanism sends this fatal message. User is returned to PRIMOS command level. [E\$NEXP]

- Beginning of file. File System

An attempt was made to access locations before the beginning of the file. (PRWF\$\$, RDEN\$\$, SGDR\$\$) [E\$BOF]

- \*\*\*\*BN n R-mode function

Device error in REWIND command on FORTRAN logical unit n.

- Buffer too small. File System

Buffer as defined is not large enough to accomodate entry to be read into it. (RDEN\$\$) [E\$BFTS]



- Cannot access like reference.

File System

Object specified in the -LIKE option of SET\_ACCESS could not be accessed for some reason. (AC\$LIK) [E\$LRNA]

- Category protects MFD.

File System

An attempt was made to delete an access category which protects the MFD. The MFD must be removed from the category before the category can be deleted. (CAT\$DL) [E\$CPMF]

- Command line more than 160 characters.

PRIMOS

An illegal command line has been received. The command is not executed, and the user is returned to PRIMOS command level. [E\$TRCL]

- Command line truncated.

PRIMOS

A command line has exceeded the limit of 160 characters.

- Concealed stack overflow.

PRIMOS

System error. (Generally sent by the condition mechanism.) [E\$CSOV]

- Crawlout unwind failed.

PRIMOS

System error. (Generally sent by the condition mechanism.) [E\$CRUN]

- \*\*\*\* DATAN - BAD ARGUMENT

V-mode function

The second argument in the DATAN2 function is zero.

- \*\*\*\*DE

R-mode function

The exponent of a double-precision number has overflowed.

- The device is in use.

File System

An attempt was made to ASSIGN a device currently assigned to another user. [E\$DVIU]

- Device not assigned.

File System

An attempt was made to perform I/O operations on a device before assigning that device. [E\$NASS]



- Device is not started. File System

An attempt was made to access a disk not physically or logically connected to the system. If disk must be accessed, the system operator must start it up. [E\$DNS]

- \*\*\*\* DEXP - ARGUMENT TOO LARGE V-mode function

The argument of the DEXP function is too large; that is, it will give a result outside the legal range.

- \*\*\*\* DEXP - OVERFLOW\*UNDERFLOW V-mode function

An overflow or underflow condition occurred in calculating the DEXP function.

- The directory is damaged. File System

UFD has become corrupted. (All file system subroutines) [E\$BUFD]

- The directory is not empty. File System

An attempt was made to delete a non-empty directory. (SRCH\$\$, FIL\$DL) [E\$DNTE]

- Directory still contains access categories. File System

You have tried to convert an ACL directory to a password directory, but the directory still contains one or more access categories, which is not allowed. (AC\$RVT) [E\$CATF]

- Directory still contains ACL subdirectories. File System

You have tried to convert an ACL directory to a password directory, but the directory still contains one or more ACL subdirectories, which is not allowed. (AC\$RVT) [E\$ADRF]

- Directory too large. File System

An attempt has been made to add more than 64K to a UFD. (CREA\$\$, SRCH\$\$, AC\$SET, AC\$CHG) [E\$FDL]

- Disk format does not support this revision of PRIMOS. File System

An attempt was made to convert a password directory to an ACL directory on a pre-Rev. 19 disk. (AC\$SET, AC\$DFT) [E\$ST19]



- DISK FULL

Old file call

There is no more room for creating/extending any type of file on disk.  
[DJ]

- Disk has been shut down.

File System

The disk has been shut down. [E\$SHDN]

- The disk is full.

File System

There is no more room for creating/extending any type of file on disk.  
(CREA\$\$, PRWF\$\$, SRCH\$\$, SGDR\$\$) [E\$DKFL]

### Note

Space may be made available. Use the PRIMOS commands ATTACH, LD, and DELETE to remove files which are no longer needed.

- Disk I/O error

File System

A read/write error was encountered in accessing disk. Returns immediately to PRIMOS level. Not correctable by applications programmer. If this happens, notify your System Administrator. (All file system subroutines) [E\$DISK]

- Disk is write-protected.

File System

An attempt has been made to write to a disk which is WRITE-protected. (All file system subroutines) [E\$WTPR]

- DK ERROR

Old file call

A read/write error was encountered in accessing disk. Try again; if you receive the same error, contact your System Administrator. [WB]

- \*\*\*\*DL

R-mode function

Argument was not greater than zero in DLOG or DLOG2 function.

- \*\*\*\* DLOG\*DLOG2 - ARGUMENT <=0

V-mode function

Argument not greater than zero was used in DLOG or DLOG2 function.



- \*\*\*\*DN *n* R-mode function  
Device error (end of file) on FORTRAN logical unit *n*.
  
- \*\*\*\* DSIN\*DCOS - ARGUMENT RANGE ERROR V-mode function  
Argument outside legal range for DSIN or DCOS function.
  
- \*\*\*\* DSQRT - ARGUMENT <0 V-mode function  
Negative argument in DSQRT function.
  
- \*\*\*\*DT R-mode function  
Second argument is zero in DATAN2 function. (D\$22)
  
- DUPLICATE NAME Old file call  
Attempt to create/rename a file with the name of an existing file.  
[CZ]
  
- \*\*\*\*DZ R-mode function  
Attempt to divide by zero (double-precision).
  
- End of file. File System  
Attempt to access location after the end of the file. (PRWF\$\$, RDEN\$\$, SGDR\$\$) [E\$EOF]
  
- \*\*\*\*EQ R-mode function  
Exponent overflow. (A\$81)
  
- Entry is an access category. File System  
An attempt was made to open an access category, as for example, with the command SLIST. (SRCH\$\$) [E\$IACL]
  
- \*\*\*\*EX R-mode function  
Exponent function value too large in EXP or DEXP function.



- \*\*\*\* EXP - ARGUMENT TOO LARGE V-mode function

The argument of the EXP function is too large, i.e., it will give a result outside the legal range.

- \*\*\*\* EXP - OVERFLOW V-mode function

Overflow occurred in calculating the EXP function.

- Fatal error in crawlout. PRIMOS

System error. [E\$CRWL]

- \*\*\*\*\*FE R-mode function

Error in FORMAT statement. FORMAT statements are not completely checked at compile time. (F\$IO)

- File in use. File System

The message may have two causes. First, you may have attempted to open a file already opened by you or by some other user. Second, you may have attempted to set a quota on a directory that currently has none and to which someone is attached or in which someone has a file open. (SRCH\$\$) [E\$FDEL]

#### Note

At Rev. 18 and above, FUTIL no longer closes open file units when it is invoked. Therefore, command files which depend on FUTIL to close units may receive "File in Use" or "File Open on Delete" messages. To avoid this message, close files explicitly, using the CLOSE command.

- File is delete-protected. File System

The delete protect switch has been set. (SRCH\$\$, FIL\$DL) [E\$DLPR]

- The file is too long. File System

Attempt made to increase size of segment directory beyond size limit. (SGDR\$\$) [E\$FITB]



- File open on delete File System  
Attempt made to delete a file which is open. (SRCH\$\$, FIL\$DL) [E\$FDEL]

Note

At Rev. 18 and above, FUTIL no longer closes open file units when it is invoked. Therefore, command files which depend on FUTIL to close units may receive "File in Use" or "File Open on Delete" messages. To avoid this message, close files explicitly, using the CLOSE command.

- \*\*\*\*FN n R-mode function  
Device error in BACKSPACE command on FORTRAN logical unit n.
- \*\*\*\* F\$BN - BAD LOGICAL UNIT V-mode function  
FORTRAN logical unit number out of range.
- \*\*\*\* F\$IO - FORMAT ERROR V-mode function  
Incorrect FORMAT statement. FORMAT statements are not completely checked at compile time.
- \*\*\*\* F\$IO - FORMAT\*DATA MISMATCH V-mode function  
Input data does not correspond to FORMAT statement.
- \*\*\*\* F\$IO - NULL READ UNIT V-mode function  
FORTRAN logical unit for READ statement not configured properly.
- F\$IOBF overflow. FORTRAN I/O  
You are trying to input or output more data than the internal buffer in the I/O subroutines can hold. A discussion of possible solutions appears in the Subroutines Reference Guide. [E\$BKOV]



- Format/data mismatch.

FORTRAN I/O

A program contains a format/data mismatch; that is, format is numeric and data is alpha. If you are using formatted I/O, the data item you are inputting or outputting is not the type (such as integer, real, or character) that you specified in your format statement. If you are using list-directed input, any character data must be given in quotation marks. You may also have tried to read a real number into an integer variable. [E\$FDM]

- \*\*\*\*II

R-mode function

Exponentiation exceeds integer size. (E\$11)

- \*\*\*\* I\*\*I - ARGUMENT ERROR

V-mode function

Exponentiation exceeds integer size.

- Illegal access mode.

File System

The access portion of an access pair contains an unknown access mnemonic. (AC\$SET, AC\$CHG) [E\$BMOD]

- Illegal identifier.

File System

The identifier portion of an access pair contains an illegal user id or group id. (AC\$SET, AC\$CHG) [E\$BID]

- ILLEGAL INSTRUCTION AT octal-location

R mode

An instruction at octal-location cannot be identified by the computer. This message appears only with R mode programs compiled and loaded on pre-17 systems.

- Illegal name.

File System

Illegal name specified for a file or UFD. (CREA\$\$, SRCH\$\$) [E\$BNAM]

- Illegal remote reference.

File System

Attempt to perform network operations by user not on network. [E\$IREM]

- Illegal treename.

File System

The string specified for a treename is syntactically incorrect. [E\$ITRE]



- \*\*\*\*IM R-mode function

Overflow or underflow occurred during a multiply. (M\$11, E\$11)

- filename IN USE. Old file call

Attempt made to open a file already opened, or to close/delete a file opened by another user, etc. [SI]

- Incorrect access control list format. File System

ACL specified in SET\_ACCESS or EDIT\_ACCESS was not in proper format. This usually results from an omitted colon between the identifier and the access rights. (AC\$SET, AC\$CHG) [E\$BACL]

- Incorrect version number. File System

A version number was passed to an ACL routine that was not recognized. (AC\$SET, AC\$CHG, AC\$LST, GETID\$) [E\$BVER]

- Insufficient access rights. PRIMOS

User does not have necessary access rights to file system object, or to perform the action desired. [E\$NRIT]

- Invalid argument to command. PRIMOS

A command has been issued with an illegal argument. The command is not executed. [E\$BARG]

- Invalid segment number. File System

Attempt made to access segment number outside valid range. [E\$BSGN]

- \*\*\*\*I/O error on logical unit <n> PRIMOS

This FORTRAN error message is usually followed by a second message that gives more precise information on the problem. Two points to remember are:

- FORTRAN's method of identifying "logical units" does not necessarily match the unit numbers given by the STATUS UNITS command;
- FORTRAN may not consider a file unit "open" unless it is open in the needed mode. (For example, a file opened for reading only would still be considered closed for writing.)



- I/O error or device interrupt.

PRIMOS

An external device has generated an interface line defined to cause an interrupt of the user program. [E\$IEDI]

- \*\*\*\*LG

R-mode function

Argument not greater than zero in ALOG or ALOG10 function.

- Like reference not found.

File System

A reference used with the -LIKE option of SET\_ACCESS cannot be located. (AC\$LIK) [E\$LRNF]

- Max number of users exceeded.

PRIMOS

The maximum allowable number of users are already using the system. (This may mean that the operator has used the MAXUSR command to decrease the number of users temporarily.)

- Maximum remote users exceeded.

File System

No more users may access the network. [E\$IMRU]

- Maximum quota exceeded.

File System

You have tried to store records in a directory that will make it exceed the maximum number of records allowed for the directory. [E\$MXQB]

- Name is too long.

File System

Length of name in argument list exceeds 32 characters. [E\$NMLG]

- Network error detected.

PRIMENET

An error has occurred in PRIMENET while attempting to process a remote request. [E\$NETE]

- No driver for device.

FORTRAN I/O

You are trying to use a device for which IOCS does not have a driver (that is, a subroutine that interfaces with a device). [E\$NDFD]



- No information.

File System

You have tried to access a file system object and you could not. Some common reasons include: insufficient access rights, non-existent object, and wrong type. This message does not reveal whether the specified object exists or not. [E\$NINF]

- No phantoms are available.

PRIMOS

An attempt has been made to spawn a phantom. All configured phantoms are already in use. (PHNTM\$) [E\$NPHA]

- No NPX slaves available.

PRIMENET

There has been a remote reference to FAM II and there are no slave processes available on the remote system. [E\$NSLA]

- No on-unit found.

Condition mechanism

Condition mechanism cannot take action. User is returned to PRIMOS command level. [E\$NOON]

- No room.

File System

An attempt has been made to add to a table of assignable devices with a DISKS or ASSIGN AMLC command and the table is already filled. [E\$ROOM]

- No room in output buffer.

PRIMENET

An error has occurred in PRIMENET while attempting to process a remote request. [E\$NROB]

- No timer.

File System

Clock not started. System error. [E\$NTIM]

- NO UFD ATTACHED

Old file call

User not attached to a UFD. Usually occurs after attempt to attach with a bad password in pre-Rev. 19.0 systems. [AL, SL]

- No UFD attached.

File System

User not attached to a UFD. Usually occurs after attempt to attach with a bad password in pre-Rev. 19.0 systems. (ATCH\$\$, CREA\$\$, GPAS\$\$, SATR\$\$, SRCH\$\$) [E\$NATT]



- Not a file or a directory.

File System

An attempt was made to protect an access category with an ACL. Only files, directories, and segment directories can be protected by an ACL. (AC\$SET, AC\$CHG, AC\$LIK, AC\$CAT) [E\$NTFD]

- Not a quota disk.

File System

An attempt was made to set or list a quota on a pre-Rev. 19 disk. (Q\$SET) [E\$NOQD]

- Not a segment directory.

File System

Attempt to perform segment directory operations on a file system object that is not a segment directory. (SRCH\$\$) [E\$NTSD]

- NOT A UFD.

Old file call

Attempt to perform UFD operations on a file which is not a UFD. [AR]

- Not a UFD

File System

Attempt to perform UFD operations on a file which is not a UFD. (ATCH\$\$, GPAS\$\$, SRCH\$\$). [E\$NTUD]

- Not an access category.

File System

This is not an access category. (AC\$CAT) [E\$NCAT]

- Not an ACL directory.

File System

You have attempted to use an ACL command (other than SET\_ACCESS) on a password directory. [E\$NACL]

- device-name NOT ASSIGNED

PRIMOS

User program has attempted to access an I/O device which has not been assigned to the user by a PRIMOS command.

- filename NOT FOUND

Old file call

File specified in subroutine call not found. [AH, SH]



- Not found. filename File System  
 File specified in subroutine call not found. (Any file system subroutine). [E\$FNTF]
- Not found in segment directory. File System  
 Filename specified in subroutine call not found in specified segment directory. (SRCH\$\$, SGDR\$\$, SRSFX\$) [E\$FNIS]
- NULL READ UNIT PRIMOS  
 Program has attempted to read with a bad unit number. This may be caused by the program overwriting itself (array out of bounds).
- Object is category-protected. File System  
 An attempt was made to use EDIT\_ACCESS on an object currently protected by an access category. (AC\$CHG) [E\$CTPR]
- Object is default-protected. File System  
 An attempt was made to use EDIT\_ACCESS on an object that is currently default protected. (AC\$CHG) [E\$DFPR]
- Operation illegal on directory. File System  
 User has tried to perform an operation on a directory that is not allowed (such as editing). [E\$DIRE]
- Operation partially blocked. PRIMENET  
 An error has occurred in PRIMENET while attempting to process a remote request. [E\$PRTL]
- Operation unsuccessful. PRIMENET  
 An error has occurred in PRIMENET while attempting to process a remote request. {E\$NSUC}
- \*\*\*\*PA n R-mode function  
 PAUSE statement n (octal) encountered during program execution.



- Parent not an ACL directory.

File System

An attempt was made to convert a password protected directory to an ACL directory, but the parent of the directory being converted was a password directory. (AC\$SET, AC\$DFT) [E\$PNAC]

- \*\*\*\* PAUSE n

V-mode function

PAUSE statement n (octal) encountered during program execution.

- PIO instruction did not skip.

PRIMOS

An invalid PIO sequence has been issued (for example, attempting to start DMX activity before loading the DMX channel address register). [E\$DNSK]

- Pointer mismatch found.

File System

Internal file pointers have become corrupted. No user remedial action possible. System Administrator must correct. [E\$PTRM]

- Priority ACL not found.

File System

No priority ACL exists for the partition specified in a LIST\_PRIORITY\_ACCESS command. (PA\$LST) [E\$PANF]

- Procedure not found.

PRIMENET

An attempt has been made to call a PRIMOS routine that does not exist on a remote system. This can happen when there is a call from a new to an old Rev. and is generated when a slave takes a linkage fault. [E\$PNTF]

- Program halt at segment no./word no.

R mode and 64V mode

Program control has been lost. The program has probably overwritten itself or the load was incomplete (R-mode).

- PRWFIL BOF

Old file call

Attempt by PRWFIL subroutine to access location before beginning of file. [PG]



- PRWFIL EOF Old file call

Attempt by PRWFIL subroutine to access location after end of file.  
[PE]
- PRWFIL POINTER MISMATCH Old file call

The internal file pointers in the PRWFIL subroutine have become corrupted.
- PRWFIL UNIT NOT OPEN Old file call

The PRWFIL subroutine is attempting to perform operations using a PRIMOS file unit number on which no file is open.
- PTR MISMATCH File System

Internal file pointers have become corrupted. No user remedial action possible. Consult your System Administrator. (ATCH\$\$, CREA\$\$, GPAS\$\$, PRWF\$\$, RDE\$\$, SATR\$\$, SRCH\$\$, SGDR\$\$)
- Quota set below current usage. File System

You have set a quota on a directory, but the directory already contains more records than the quota allows. This is a warning. (Q\$SET)  
[E\$QEXC]
- The remote line is down. PRIMENET

The network connection between the local system and the remote system cannot be established. [E\$RLDN]
- Remote system not up. PRIMENET

An attempt has been made to access a remote system that is down.  
[E\$RSNU]
- Requires receive enabled. PRIMOS

An attempt has been made to send a message without receive being enabled. [E\$NRCV]
- \*\*\*\*RI R-mode function

Argument is too large for real-to-integer conversion. (C\$12)



- \*\*\*\*RN n R-mode function

Device error or end of file in READ statement on FORTRAN logical unit n.
- \*\*\*\*SE R-mode function

Single precision exponent overflow.
- SEG-DIR ER Old file call

Error encountered in segment directory operation. [SQ]
- Segment directory error. PRIMOS

Error encountered in segment directory operation. [SQ] [E\$SDER]
- Segment directory unit not open. File System

Attempt has been made to reference a segment directory which is not open. (SRCH\$\$) [E\$SUNO]
- Semaphore overflow. File System

System error. [E\$SEMO]
- \*\*\*\* SIN/COS - ARGUMENT TOO LARGE V-mode function

Argument too large for SIN or COS function.
- Slave validation error. PRIMENET

Either you are trying to access a remote system that requires user validation, and there was no ADD\_REMOTE\_ID command issued (to establish a remote id); or a previous ADD\_REMOTE\_ID command established a user id, project id, or password that was invalid on the remote system. [E\$SVAL]
- \*\*\*\*SQ R-mode function

Negative argument in SQRT or DSQRT function.
- \*\*\*\* SQRT - ARGUMENT<0 V-mode function

Negative argument in SQRT function.



- \*\*\*\*ST n R-mode function  
 STOP statement n (octal) encountered during program execution.
- Stack overflow in crawlout. PRIMOS  
 System error. [E\$CROV]
- \*\*\*\* STOP n V-mode function  
 STOP statement n (octal) encountered during program execution.
- \*\*\*\*SZ R-mode function  
 Attempt to divide by zero (single-precision).
- System console command only. PRIMOS  
 The command issued must be from the system console. [E\$SCOM]
- Top-level directory not found or inaccessible. File System  
 An attempt to attach to a top-level directory has failed because the UFD does not exist, because PRIMOS cannot reach the system where it does exist (if a remote system is down), or because you do not have the right (use, or U, access) to attach to it. You remain attached to the previous current directory. (AT\$ANY, AT\$) [E\$NFAS]
- Unable to find fault frame. Condition mechanism  
 A call was made to CNSIG\$, but CNSIG\$ could not find that any condition had been raised.
- UNIT IN USE Old file call  
 Attempt to open file on PRIMOS file unit already in use. [SI]
- Unit in use. File System  
 Attempt to open file on PRIMOS file unit already in use. (SRCH\$\$)  
 [E\$UIUS]



- UNIT NOT OPEN

Old file call

Attempt to perform operations with a file unit number on which no file has been opened, or which is opened in the wrong mode (for example, a read to a unit open only for writing). [PD, SD]

- Unit not open.

File System

Attempt to perform operations with a file unit number on which no file has been opened, or which is opened in the wrong mode (for example, a read to a unit open only for writing). (PRWF\$\$, RDEN\$\$, SRCH\$\$, SGDR\$\$) [E\$UNOP]

- UNIT OPEN ON DELETE

Old file call

Attempt to delete file without having first closed it. [SD]

- \*\*\* Unknown addressee.

PRIMOS

The user id you have specified to receive a message is not the id of a logged-in user. [E\$UADR]

- \*\*\* User usernumber busy, please wait.

PRIMOS

The user to whom you have sent a message already has a deferred message waiting. Only one deferred message is allowed. You must send your message again. [E\$SUBSY]

- \*\*\* User usernumber not receiving now.

PRIMOS

The receive state of the user to whom you wish to send a message is either DEFER or REJECT. [E\$UNRV]

- User unable to receive messages.

PRIMOS

An attempt was made to send a message to another user that for various reasons was not successful. [E\$UDEF]

- Warm start occurred.

PRIMOS

Any device connected to GPPI has received a reset signal. [E\$WMST]



- Wrong file type.

FORTRAN I/O

You are trying to use direct access I/O when you declared the file to be sequential in the OPEN statement, or vice versa. [E\$WFT]

- \*\*\*\*WN n

R-mode function

Device error or end-of-file in WRITE statement on FORTRAN logical unit n.

- \*\*\*\*XX

R-mode function

Integer argument >32767.



BATCH WARNINGS AND MESSAGES

- Bad \$\$ command

(Fatal) A command file was submitted using the JOB command that had a \$\$ line other than the \$\$ JOB line as the first non-comment line. The command file should be changed so that the "\$\$" line is legal. The use of \$\$ is reserved for future expansion by BATCH.

- (Changes made)

(Response) The changes specified in a JOB -CHANGE operation have been made. If the job is initiated after the changes are made, then it will execute with the specified changes in place. The job status will be displayed after the above message is typed out.

- Command or CPL file required as first argument on submission.

(Fatal) The JOB command was given with job options (such as -HOME, -PRIORITY, -CPTIME, etc.) but no command or CPL file was seen before those options. The syntax is "JOB pathname [options]".

- Cpu limit must be specified

(Fatal) The queue referred to by a -QUEUE option during job submission is defined such that the -CPTIME option is a required parameter (that is, default CPU limit for that queue is greater than the maximum CPU limit for that queue). The job should be resubmitted with the -CPTIME option specified. To determine the maximum limits for queues, use BATGEN -DISPLAY.

- Elapsed time limit must be specified.

(Fatal) The queue referred to by a -QUEUE option during job submission has a default elapsed time limit greater than its maximum time limit. Resubmit the job with the -ETIME option specified.

- End of line.

(Fatal) One of the Batch programs was expecting to find more information on the command line, but end-of-line was found instead. The message will generally contain more information on what was expected. Re-enter the command with the additional requested information.



- End of line. Illegal <option> argument

(Fatal) One of the job parameter options specified on the JOB command line had no argument. The information required by that option should be supplied when the command is re-entered.

- File has no non-comment lines. <filename> (JOB)

User has tried to submit a CPL program or command file that contains no commands. (The file either is empty or is made up entirely of comments.)

- Home ufd required.

(Fatal) The -HOME option was not present on the JOB or the (optional) \$\$ JOB line during submission, and the program was unable to determine the home attach point of the submitting job. Resubmit the job, and include the -HOME option followed by the absolute pathname of the directory where the job is to execute. If the pathname cannot fit, use a shorter version of it when you resubmit the command file, after editing the file to include an "ATTACH" command that descends the remaining sub-UFDS to reach the destination.

- Home=<pathname>

(Response) During job submission, the -HOME option was not specified on the command line or in the command file (\$\$ JOB), so the JOB command determined the home attach point of the submitting job. This message is typed out to remind the user that the -HOME option was not specified. However, the job did successfully submit.

- Illegal -CHANGE option.

(Fatal) The options -QUEUE and -PRIORITY are illegal during a -CHANGE operation using the JOB command, as the queue and queue priority of a job cannot be changed. Cancel or abort the job, and resubmit it into the appropriate queue with the desired queue priority.

- Illegal combination. <option>

(Fatal) A job parameter (such as -ACCT, -HOME or -QUEUE, etc.) was specified on the same JOB command line as an option to perform an action (such as -CANCEL, -DISPLAY, -ABORT, etc.). Use separate JOB commands to perform separate functions.



Note

This message can also result from giving the -FUNIT option for a CPL program. CPL files cannot specify FUNTIS.

- Illegal limit.

(Fatal) The parameters supplied to the -CPTIME or -ETIME options during job submission or changing were not legal limits; that is, they were less than or equal to 0; they were not legal decimal numbers; or they were not the string "None". Re-enter the command with legal limits.

- Illegal name.

(Fatal) One of the Batch programs was expecting a name or command, but it read an unquoted token beginning with a dash ('-'), indicating that an option was present.

- Illegal number. <text> (JOB)

(Fatal) The argument for the -FUNIT or -PRIORITY option during job submission using the JOB command was not a legal decimal number. Re-enter the command line with legal numeric parameters.

- Illegal option.

(Fatal) One of the Batch programs was expecting an option, that is, an unquoted token beginning with a dash ('-'). Re-enter the command line with a legal format.

- Illegal queue name. <text> (JOB)

(Fatal) The queue name specified after a -QUEUE option while submitting or changing a job did not comply with queue name format rules. Use BATGEN -STATUS or -DISPLAY to determine the names of legal queues.

- Incorrect user-name.

(Fatal) A command file was submitted using the JOB command that had a \$\$ JOB line as the first non-comment line, but the user-name specified after the "JOB" specifier did not match the user-name of the submitting user. Edit the command file and change the user-name in the \$\$ JOB line to the user-name of the submitter.



- \*\*\*Invalid batch database, please contact your system administrator

(Severe) The running job detected an error (such as disk failure, pointer mismatch, or misprotected file) in the Batch system database. It will flag the database as invalid. Notify the System Administrator, who has the responsibility for re-initializing the database (or running FIXBAT or FIXRAT as the case may be). The BATCH and JOB commands will be inoperative until the situation is resolved.

- <nn> is out of range. <option>

(Fatal) The numbers supplied as parameters to the -FUNIT or -PRIORITY options during job submission/changing were out of range. The range for -FUNIT is from 1 to 126; that for -PRIORITY is from 0 to 9. The job should be resubmitted or changed with legal -FUNIT and -PRIORITY values. Note that the system may be configured to have fewer than 126 units per user at cold-start, and the -FUNIT argument will be limited to the maximum configured unit number.

- ?Job <extrnam>(<intrnam>) <status>.

(Warning) An attempt was made to perform an operation on a job using the JOB command that could not be performed because of its status: for example, trying to restart a completed job.

- Job name required.

(Fatal) The options -CHANGE, -CANCEL, -ABORT, -RESTART, -HOLD and -RELEASE all require a job identifier (internal or external name). Re-enter the command with the job id. (For example: "JOB C.TOP -HOLD", "JOB #10032 -ABORT").

- Job not found.

(Fatal) The job referred to in a JOB command such as -CHANGE, -CANCEL, -ABORT, -RESTART, -HOLD or -RELEASE, could not be found by searching the active jobs list. This could mean one of three things: that no job exists with that name; that all jobs that have that name are not active jobs (that is, have completed, aborted or been cancelled); or that a job exists with that external name, but the user making the request is not the same user that originally submitted the job.

- (Job no longer restartable)

(Response) A JOB -CANCEL was performed on an executing job. The job itself is not cancelled; it has been flagged as being unrestartable (that is, a -RESTART will abort the job but not restart it).



- (Job not restartable)

(Warning) A JOB -RESTART was performed on a job that had been flagged as unrestartable. An attempt will be made to abort the job.

- (Job restarted)

(Response) A JOB -RESTART was performed on a job, and the job is restartable. Although an error message may appear after this message, the job will generally be restarted unless a JOB -CANCEL or JOB -CHANGE -RESTART NO is given. The error message "Insufficient access rights" may follow this message if the user is logged in as SYSTEM and has restarted another user's job from a user terminal (rather than from the supervisor terminal). If the process recently logged out, the error messages "Insufficient access rights" or "Not found" may be returned.

- \*\*\* Jobs are not being processed at this time.

(Severe) If followed by "\*\*\* Please contact your system administrator immediately", it indicates that the Batch database has not been initialized, or that something has happened to it (such as a disk head crash). If followed by "\*\*\* Please try again later", it indicates that while the database is still valid, the Batch monitor was logged out using a method other than "BATCH -STOP", and will verify the validity of the database when it is started up. Either way, the user will be immediately returned to command mode (that is, the operation the user attempted will not be performed). This message can be typed out by the BATCH or the JOB commands when they start running.

- Multiple jobs with this name (use internal name).

(Fatal) A reference was made to a job using a filename in the JOB command, and there were at least 2 such jobs belonging to the user making the reference that were active. The job id must be used in this case. Use JOB -STATUS to determine the filenames and job ids of all jobs belonging to the user issuing the command.

- Multiple occurrence.

(Fatal) An option was specified twice during job submission or job changing (example: JOB C\_TEST -HOME HERE -HOME THERE) on either the JOB or \$\$ JOB line. (If the option is specified once on the JOB line and once on the \$\$ JOB line, no error will result and the parameter on the JOB line will take precedence.) Re-enter the command, specifying each option only once.



- Must be first option.

(Fatal) The options -CHANGE, -CANCEL, -ABORT, -RESTART, -STATUS, -DISPLAY, -HOLD and -RELEASE must be the first option on the JOB command line (after a sometimes optional job identifier). Use the JOB command several times to perform several operations.

- No active jobs [named "<jobname>"] for user <username>.

(Response) There are no jobs belonging to that user that are waiting, held, or executing. The jobname is output if a jobname was specified for the -DISPLAY or -STATUS command; otherwise it is omitted.

- No batch jobs.

(Response) This message is typed out by a BATCH -STATUS command and indicates that there are no active Batch jobs.

- No job changes specified.

(Fatal) The -CHANGE option was given to the JOB command, but no actual changes were specified on the command line. Specify changes to be made after the -CHANGE option.

- No longer executing.

(Fatal) A JOB -ABORT or JOB -RESTART was performed on a job that had execution status, but by the time the execution file was read in to determine the user number of the process, it had disappeared. If the message "(Job restarted)" had been typed out, then the job would be restarted.

- No queue available for job.

(Fatal) A job was submitted using the JOB command that did not use the -QUEUE option to specify the queue to which it was to be submitted, and no suitable queue could be found. Suitability for a queue includes CPU and elapsed time limits being within the confines of the queue, queue being unblocked, etc. Use the BATGEN -STATUS or -DISPLAY command to yield a list of legal queues and their status.

- No queues have waiting or held jobs.

(Response) A BATCH -DISPLAY command was issued, and there were no queues that had any waiting or held jobs in them. A queue may have one executing job in it, but an executing job is not considered a waiting or held job.



- No running jobs.

(Response) A BATCH -DISPLAY command was issued, and there were no jobs that were currently running. However, it is possible to have no running jobs and get to have jobs waiting, even when the monitor is running and there are free phantoms. This condition can occur because there is always a small amount of time between the submittal of a job and the execution of a job.

- Not an absolute treename.

(Fatal) You have specified a relative pathname (beginning with ">") as the home UFD with the -HOME option of the JOB command. Re-submit the job, giving an absolute (full) pathname after the -HOME option.

- Not your job.

(Fatal) A reference was made to a job using an internal name in the JOB command, and the referenced job did not belong to the user making the reference. Use "JOB -STATUS" to obtain a list of all jobs belonging to the user making the request.

- Null home ufd.

(Fatal) The home UFD specified with the -HOME option during submission using the JOB command (or changing of job parameters) was a null string. Re-submit the job with an absolute pathname after the -HOME option.

- Please stand by.

(Response) This message and others like it ("File in use, please stand by") will be output if the program being run is trying to gain access to a file that is in use for more than 5 seconds. After 30 seconds, the "File in use..." message will be output, and after 60 seconds, the message "Timeout of 60 seconds has occurred" will be output and the program will "give up". Usually this will result in a fatal error, as it could indicate that system security is broken.

- Please wait.

(Response) This message asks that the user be patient because the program he is running has been locking up the Batch database too long and is not allowing other processes to have access to it. It is not a fatal error. It generally is output only when a system is heavily loaded, or when the current process has a very low priority and does not run frequently.



- Queue blocked.

(Fatal) The queue referred to by a -QUEUE option during job submission is currently blocked to new submissions. Try it again later, or use another queue.

- Queue deleted.

(Fatal) The queue that the job was being submitted to was present when it was first checked out, but by the time the command file had been copied and some other activities had taken place, the queue had been deleted. The job should be resubmitted to a different queue.

- Queue does not exist.

(Fatal) The -QUEUE option on the JOB command line or the (optional) \$\$ JOB line referred to a queue that either did not exist or was in the process of being deleted ("flagged for deletion"). The BATGEN -STATUS or -DISPLAY command should provide a list of currently available queues and their status.

- Queue full.

(Fatal) There are already 10,000 jobs (whether active or inactive) in the queue to which the job is being submitted. The queue must be deleted and re-created before more jobs can be submitted to it. The system administrator should be asked to do this. Meanwhile, if any other queues are available, they can be used instead by the user.

- Register setting.

(Fatal) Register settings are illegal in the Batch subsystem (except as part of a submitted command file). Re-enter the command line without the register setting.

- Searching for free command file, please stand by.

(Response) This and other messages like "Queue is in heavy use...please stand by" mean that many users are submitting command files at once. The situation should resolve itself in a short amount of time.



- Specified value is out of range.

(Fatal) The -CPTIME or -ETIME option specified during job submission or a -CHANGE operation is greater than the maximum allowed by the queue to which the job was submitted. This message will be preceded by a message indicating the maximum limit for that queue ("Cpu limit is xx" or "Elapsed time limit is xx"). If the limits cannot be lowered and the job successfully run, then try a queue with higher limits.

- Syntax error. Register settings are illegal

(Warning) This message is output if end-of-line is expected and a register setting is found instead. Re-enter the command without register settings.

- <text> seen when end-of-line expected.

(Fatal) text was seen when there should have been no more text (end of line). The command will be ignored and the user will be returned to PRIMOS level.

- This job cannot be restarted.

(Response) Output by a JOB -DISPLAY command if the job being displayed has had a JOB -CANCEL done to it while it was executing, or was submitted with the -RESTART NO option. Any -RESTARTs done to the job will abort the job (if they succeed), but the job will not be restarted.

- (This job has already executed nn time(s)).

(Response) Output by a JOB -DISPLAY command if the job being displayed is executing and has already been executed. This is the result of a JOB -RESTART being done on that job, or a system cold-start after being brought down while the job was executing.

- Too many options.

(Fatal) At least two options were entered that conflicted with each other, such as JOB -DISPLAY -CHANGE or JOB C TEST -ABORT -CANCEL. Use separate JOB commands to perform separate operations.

- Unknown option.

(Fatal) An option was entered to the BATCH or JOB command that was not recognized.



- Warning: jobs are not being processed at this time.

(Response) The Batch monitor is not running. No submitted jobs will be executed until it has been started up. The operation requested will then be performed.

- Your job, job-id, could be in queue queuename, but may not execute due to the following error.

(Fatal) The appropriate error message is printed after this message, followed by an "ER!" prompt. Command input is suspended.



CONDITIONS FLAGGED BY THE CONDITION MECHANISM

When the system default on-unit is activated, it prints the following message at the user's terminal.

```
Error:  condition "condition" raised at "address"
        [further information]
```

condition names the condition invoked. address gives the location in the form:

segment no. (ring no.)/word no. (bit offset)

The bit offset number does not always appear. The following example shows a possible message generated by the condition mechanism for the SIZE condition:

```
SIZE raised at 4000(3)/1004
(in float to fixed conversion)
```

```
ERROR raised at 4000(3)/1004
(no on-unit for SIZE)
```

```
Now at command level 5. To release use RLS. (listen_)
ER!
```

The following list describes the conditions that may result in condition messages.

- ACCESS\_VIOLATION\$

The process has attempted to perform a CPU instruction which has violated the access control rules of the processor. No information is readily available to differentiate between write violation, read violation, execute violation and gate violation.

- ALARM\$

When the real time watchdog timer expires, the ALARM\$ condition is raised. Only those people who have explicitly set the real time watchdog timer will get this notice. This is an informational notice. You set the timer because you want to be notified that a certain length of time has passed.



- ANY\$

An activation's on-unit for ANY\$ is invoked if that activation does not have a specific on-unit for the condition that was raised. The condition frame header for the condition ANY\$ will describe the original condition directly; there is no separate condition frame header for the condition ANY\$ unless ANY\$ has been explicitly raised by a call to SIGNL\$ (not a recommended practice).

- AREA

This condition is raised when a storage area has been damaged, or when the target area for an attempted copy from one area to another was too small. (Generally raised by full PL/I only. Not available through PL/I Subset G.)

- ARITH\$

The process has encountered an arithmetic exception fault.

The static mode default on-unit for this condition will simulate Prime 300 fault handling for arithmetic exception if the appropriate word of segment '4000 is nonzero. (See the System Architecture Reference Guide for the exact location.) If a static mode program is not in execution when the fault occurs, or if the Prime 300 vector word is 0, the standard default handler for this condition will resignal the appropriate arithmetic condition (size, fixedoverflow, etc.) with the appropriate information structure.

- BAD\_NONLOCAL\_GOTOS

The nonlocal GOTO processor has been asked to transfer control to a label whose display (stack) pointer is invalid, or whose target activation has already been cleaned up. There is also a possibility that the user's stack may have been overwritten.

- BAD\_PASSWORD\$

This condition is raised by the ATCH\$\$ primitive when attempting to attach with an incorrect password to a directory requiring a password. This condition is signalled nonreturnable in order to increase the work function of machine-aided password penetration.



- CLEANUP\$

The nonlocal GOTO processor (UNWIND\_) is in the process of invoking on-units for the condition CLEANUP\$ in each activation on the stack, prior to actually unwinding the stack. The on-unit for this condition should return, unless it encounters a fatal error. Calls to CNSIG\$ from a CLEANUP\$ on-unit have no effect.

- COMI\_EOF\$

End of file occurred on the command input file. The default on-unit prints a diagnostic message and returns to the point of interrupt.

- CONVERSION

This condition is raised when the source data for a data-type conversion contains one or more characters that are invalid for the target type. For example, nonnumeric characters appear in a character string which is to be converted to integer.

- CPU\_TIMER\$

When the CPU watchdog timer expires, the CPU\_TIMER\$ condition is raised. Only those people who have explicitly set the CPU watchdog timer will get this notice. This is an informational notice. You set the timer because you want to be notified after a certain amount of CPU time has been used.

- DAMAGED\_RAT\$

You are trying to write to a file. When the PRIMOS file system goes to the record availability table (RAT) to assign a record for your work, it finds that the RAT is damaged. This is a system-wide problem. See your System Administrator.

- ENDFILE (file)

This condition is raised when an end of file is encountered while reading a PL/I file with PL/I I/O statements. The value of the ONFILE() built-in function identifies the file involved.

The standard PL/I condition information structure is provided. The value of info.oncode\_value is undefined, and info.file\_ptr identifies the file on which end of file occurred.

The default on-unit for this condition prints a diagnostic and then resignals the ERROR condition with an info.oncode\_value of 1044.



- ENDPAGE (file)

This condition is raised when end of page is encountered while writing a PL/I file using PL/I I/O statements. The value of the ONFILE() built-in function identifies the file on which the end of page was encountered.

The standard PL/I condition information structure is provided. The value of info.oncode\_value is undefined; info.file\_ptr identifies the file in question.

The default on-unit for this condition performs a PUT SKIP on the file, and then returns.

- ERROR

This condition is a catch-all error condition defined in PL/I. The default on-unit for most PL/I-defined conditions (such as KEY) results in the ERROR condition being resignalled. Hence, the programmer has the choice of handling a more- or less-specific case of the condition.

- ERRRTN\$

A non-ring-0 call to the ring 0 entry ERRRTN was made, as the result of an ERRRTN SVC or a call to ERRPR\$ with certain values of the key.

The default on-unit for this condition simulates a call to EXIT; hence, this condition should be signalled only while executing in a static mode program.

- EXIT\$

The process has made a call to the EXIT primitive, via a direct call or an EXIT SVC. This condition should not be handled by user programs, since it is used by certain PRIMOS software to monitor the execution of static mode programs.

The default on-unit for this condition simply returns.

- FINISH

This condition is signalled before process termination. It closes any open files and returns to the point at which the condition was signalled. It is not signalled if the process is prematurely exhausted or destroyed. (Generally raised by full PL/I only. Not available through PL/I Subset G.)

The default on-unit simply returns.



- **FIXEDOVERFLOW**

This condition is detected by hardware and is raised when a fixed-point decimal or binary result is too large to fit into the hardware register or decimal field.

The standard PL/I condition information structure is provided.

- **ILLEGAL\_INST\$**

The process has attempted to execute an illegal instruction.

- **ILLEGAL\_ONUNIT\_RETURN\$**

An on-unit for some condition has attempted to return, when that has been disallowed by the procedure that raised the condition.

- **ILLEGAL\_SEGNO\$**

The process has referenced a virtual address whose segment number is out of bounds.

- **KEY (file)**

The KEY condition is raised when reading or writing a keyed PL/I file with PL/I I/O statements, and the supplied key does not exist (READ) or already exists (WRITE). The value of the ONFILE() built-in function identifies the file in question; the value of the ONKEY() built-in function contains the key in error.

- **LINKAGE\_FAULT\$**

The process has referenced through an indirect pointer (IP) which is a valid un-snapped dynamic link, but the desired entry point could not be found in any of the dynamic link tables.

- **LISTENER\_ORDERS\$**

This condition is used internally by the command loop to manage its recursion. Users should never make on-units for this condition, and user default on-units (ANY\$) should always pass this condition on by returning.



- LOGOUT\$

This condition is raised when a user or the operator is trying to force log out a process. The default on-unit logs out the process. Once signalled, the user process has approximately two minutes before being logged out.

- NAME

This condition occurs only during data-directed input. It occurs when stream assignment in a GET statement is read whose variable does not match the variable name in the data list. After execution of the on-unit, the process returns to the data-directed input as if the "bad" input were processed. (Generally raised by full PL/I only. Not available through PL/I Subset G.)

- NO\_AVAIL\_SEGS\$

The process has referenced a virtual address that refers to a segment that has not yet been created. At the moment, the system has no free page tables to assign to the segment. If the on-unit for this condition returns, the reference will be retried, with some possibility of success if this or some other process has in the meantime deleted a segment. You may also use the DELSEG command to release assigned segments, or log out (which also releases segments) and try again later.

- NONLOCAL\_GOTO\$

This condition is signalled by the PL/I nonlocal GOTO processor PLI\$NL just prior to setting up the stack unwind (and hence prior to the invocation of any CLEANUP\$ on-units). This condition exists to enable certain overseer software (such as the debugger) to be informed that the nonlocal GOTO is occurring. The default handler for this condition simply returns. When a procedure handling this condition wishes to let the nonlocal GOTO occur, it should simply return (without continue-to-signal set).

- NPX\_SLAVE\_SIGNED\$

A condition has been raised in your NPX slave running on the indicated node. The following message is printed:

Condition signalled in NPX slave on nodename  
 ERROR: Condition "condition name" raised at segment no./word no.



When the slave detects a signalled condition, it transmits to the master, which signals the condition `NPX_SLAVE_SIGNED$`. Its result is the printout of the message shown above. The slave transmits to the master almost all types of conditions signalled except the following:

```
EXIT$
FINISH
NONLOCAL_GOTO$
REENTER$
STRINGSIZE
```

These conditions are handled differently by slave's on-unit. They are returned without transmitting to the master, that is, the master side will not get the condition `NPX_SLAVE_SIGNED$`.

- `NULL_POINTER$`

The process has referenced through an indirect pointer or base register whose segment number is '7777'b3. This is considered to be a reference through a null pointer, although user software should always employ the single value '7777/0 for the null pointer.

The default on-unit for this condition resignals the `ERROR` condition with the appropriate information structure.

- `OUT_OF_BOUNDS$`

The process has referenced a page of some segment that has been defined as incapable of being referenced in any ring (i.e. no main memory or backing storage is allocated for that page, and allocation is not permitted).

- `OVERFLOW`

This condition is raised when the result of a floating-point binary calculation is too large for representation. It may occur within a register, or as a store exception. The default on-unit prints a message and signals the `ERROR` condition. User on-units may not return to the point of interrupt. However, if the default on-unit is invoked, and if the user types `START`, the register or memory location affected will be set to the largest possible single-precision floating-point number, and calculation will continue.



- PAGE\_FAULT\_ERR\$

The process has encountered a page fault while referencing a valid virtual address, but due to a disk error, the page control mechanism has not been able to load the page into main memory. If the on-unit for this condition returns, the reference will be retried, and there is some likelihood that the disk read will succeed and the reference thus be completed.

- PAGING\_DEVICE\_FULL\$

You have attempted to do work that requires an additional page in a segment and there is no room on the paging device to provide that page. You may be able to recover from this situation by issuing the DELSEG command (which releases segments, and therefore releases pages). If the condition persists, see your System Administrator.

- PAUSE\$

The process has executed a PAUSE statement in a FORTRAN program. This condition should not be handled by user programs since it is used by Prime software to ensure the proper operation of the FORTRAN PAUSE statement.

The default on-unit for this condition prints no diagnostic, but calls a new command level.

- PH\_LOGO\$

This condition is raised when a phantom which you spawned is logging out.

- POINTER\_FAULT\$

The process has referenced through an indirect pointer (IP) whose fault bit is on, but that pointer did not appear to be a valid unsnapped dynamic link. That is, reference has been made to an argument or instruction not in memory. This error condition is frequently caused by an incomplete load (unsatisfied references), or by making a subroutine or function call with too few arguments. The condition is raised when the called subroutine attempts to access one of its arguments through a faulted pointer.



- QUIT\$

The user has actuated QUIT (BREAK key or CONTROL-P) on the terminal.

The default on-unit flushes the input and output buffers of the user's terminal, prints the message "QUIT." on the terminal, and calls a new command level.

- RECORD

This condition is raised when record size is different from the variable defined in the PL/I source. (Generally raised by full PL/I only. Not available through PL/I Subset G.)

- REENTER\$

This condition is raised by the PRIMOS REENTER (REN) command and reenters a subsystem that has been temporarily suspended due to another condition (such as a QUIT\$ signal).

If the interrupted operation can be aborted, the subsystem's on-unit should perform a nonlocal GOTO back into the subsystem at the appropriate point.

If the QUIT\$ occurred during an operation that must be completed, the on-unit should set the info.start\_sw to '1'b, record the QUIT\$ request within the subsystem and return. The REN command will then execute a START command which will restart the subsystem at the point of interrupt. When the operation is complete, the subsystem should then honor the recorded QUIT\$ request.

The default on-unit returns without setting the info.start\_sw. The REN command will then print a diagnostic and return since it assumes there was no subsystem on the stack able to accept reentry.

- RESTRICTED\_INST\$

The process has attempted to execute an instruction whose use is restricted to ring 0 procedures. Certain of these instructions (in the I/O class) can be simulated by ring 0. An instruction which causes this condition to be raised could not be simulated by this mechanism.

- RINREC\_ERR\$

This condition is caused by an error in the return of a record to the disk record availability table (RAT). Run the process again. If you still get the same message, see your System Administrator.



- RO\_ERR\$

A ring-0 call to ERRPR\$ or ERRRTN has been made, as the result of some fatal error condition having been detected.

The default on-unit for this condition prints no diagnostic, but calls a new command level.

- SETRC\$

This condition is raised to indicate that a command input file is not to be aborted on an error severity (ER! prompt). A user's ANY\$ on-unit should continue to signal this condition.

- SIZE

This condition is raised when a program tries to do an arithmetic conversion and the value is too large to fit into the target data type. It can occur when converting either a floating-point number or a decimal integer to a binary integer.

- STACK\_OVF\$

The process has overflowed one of its stack segments, but the condition mechanism was able to locate a stack on which to raise this condition.

The static mode default on-unit will attempt to simulate the Prime 300 fault handling for stack overflow fault if the appropriate word of segment '4000 is nonzero. (See the System Architecture Reference Guide.) If this word is 0 or if no static mode program is in execution, the standard default handling occurs.

- STOP\$

The process has executed a STOP statement in a higher-level-language program. This condition should not be handled by user programs, as it is used by Prime software to ensure the proper operation of the STOP statement in the various languages.

The default on-unit for this condition performs a nonlocal GOTO back to the command processor which invoked the procedure which (or one of the dynamic descendants of which) executed the STOP statement.

- STORAGE

This condition occurs when your program attempts to allocate storage and none is available. (It is generally raised by full PL/I only and is not available through PLIG.)



- STRINGRANGE

One argument of the SUBSTR function is out of range of the string.

- STRINGSIZE

The target of a string assignment is too small to contain the value. The default on-unit simply returns.

- SUBSCRIPTRANGE

A subscript is out of range.

- SUBSYS\_ERR\$

This condition is used internally by the command loop to abort command input (COMINPUT) or CPL files that contain errors. Users should never make on-units for this condition, and user default on-units (ANY\$) should always pass this condition on by returning.

- SVC\_INST\$

The process has executed an SVC instruction, but the system has not been able to perform the operation. If the user is in "SVC virtual" mode, all SVC instructions result in this condition being raised.

For the case of virtual SVC's only (info.reason code of 3), the static mode default on-unit will simulate the Prime 300 fault handling for the SVC fault, if the appropriate word of segment '4000 is nonzero. If this word is 0 or if there is no static mode program in execution, the standard default handler prints a diagnostic and calls a new command level. (See the System Architecture Reference Guide for the exact location.)

- TRANSMIT

This condition occurs when data cannot be transmitted reliably between a data set and PL/I storage. (It is generally raised by full PL/I only and is not available through PL/I Subset G.)

- UII\$

The process has executed an unrecognized instruction that nevertheless caused an unimplemented instruction fault, or else the system UII handler detected an error in processing the valid UII.



- UNDEFINEDFILE (file)

This condition is raised when an OPEN statement cannot associate an input file with an existing PRIMOS file or device. The default on-unit prints a message and signals the ERROR condition.

- UNDEFINED\_GATE\$

This condition is signalled when the process has called an inner ring gate segment at an address within the initialized portion of the gate segment, but there was no legal gate at that address. This error can arise because gate segments are padded, from the last valid gate entry to the next page boundary, with "illegal" gate entries.

- UNDERFLOW

This condition is signalled when the result of the floating-point binary or decimal calculation is too small for representation. The default on-unit sets the floating-point accumulator to 0.0e0. If the underflow occurred as a store exception, the affected portion of memory is also set to 0.0e0. The default on-unit returns and the calculation proceeds, using the 0.0e0 value.

- ZERODIVIDE

This condition is signalled when a division by 0 (floating-point or fixed-point) occurs. The default on-unit prints a message and signals the ERROR condition. For compatibility with earlier versions of PRIMOS, if the condition is the result of a floating-point operation, the user may type START following the printing of the message. The default on-unit will then set the register involved to the largest possible single-precision, floating-point value and proceed with the calculation.



Section 1

The first part of the document discusses the importance of maintaining accurate records and the role of the committee in overseeing the process.

Section 2

The second part of the document details the specific procedures and guidelines that must be followed to ensure the integrity of the data collection process.

The third part of the document outlines the responsibilities of the various stakeholders involved in the project, including the committee members and the research team.

The final part of the document provides a summary of the key findings and conclusions drawn from the analysis of the data collected during the study.



# E

## Editor Command Summary

The following is an alphabetic list of each Editor (ED) command and its function. Acceptable command abbreviations are underlined. For a detailed description of all commands, see the Editor Reference Section of the New User's Guide To EDITOR and RUNOFF.

### Note

The string parameter in a command is any series of ASCII characters including leading, trailing, or embedded blanks. A semicolon terminates the command unless it appears within delimiters (as in the CHANGE, MODIFY, or GMODIFY commands) or is preceded by the escape character (^).

<u>Command</u>	<u>Function</u>
<u>APPEND</u> string	Appends <u>string</u> to the end of the current line.
<u>BOTTOM</u>	Moves the pointer beyond the last line of the file.
<u>BRIEF</u>	Speeds editing by suppressing the (default) verification responses to certain Editor commands.



<u>CHANGE</u> /string-1/string-2/[G] [n]	Replaces <u>string-1</u> with <u>string-2</u> for <u>n</u> lines. If <u>G</u> is omitted, only the first occurrence of <u>string-1</u> on each line is changed; if <u>G</u> is present, all occurrences on <u>n</u> lines are changed.
<u>DELETE</u> [n]	Deletes <u>n</u> lines, including the current line (default <u>n</u> = 1).
<u>DELETE TO</u> string	Deletes all lines up to but not including the line containing <u>string</u> .
<u>DUNLOAD</u> filename [n]	Deletes <u>n</u> lines from the current file and writes them into <u>filename</u> . (Default <u>n</u> = 1.)
<u>DUNLOAD</u> filename <u>TO</u> string	Same as <u>DELETE...TO</u> , but writes deleted lines into <u>filename</u> .
<u>ERASE</u> character	Sets erase character to <u>character</u> .
<u>FILE</u> [filename]	Writes the contents of the current file into <u>filename</u> and QUITs to PRIMOS. If <u>filename</u> is omitted, EDITOR writes into the current file and prints its name.
<u>FIND</u> string	Moves the pointer down to the first line beginning with <u>string</u> .
<u>FIND</u> ( <u>n</u> ) string	Moves the pointer down to the first line in which <u>string</u> starts in column <u>n</u> .
<u>FNAME</u>	Prints the name of the current file during an editing session.
<u>GMODIFY</u>	Allows the user to enter a string of subcommands which modify characters within a line.
<u>IB</u> string	The <u>INSERT BEFORE</u> command inserts <u>string</u> as a new line immediately before the current line.
<u>INPUT</u> { (ASR) (PTR) (TTY) }	Reads text from the specified input device: ASR (teletype paper-tape reader), PTR (high-speed paper tape reader) or TTY (terminal). Default is TTY.
<u>INSERT</u> string	Inserts <u>string</u> after the current line.
<u>KILL</u> character	Sets kill character to <u>character</u> .



<u>LINESZ</u> [n]	Changes maximum line size (that is, length) of both command lines and input lines. This length, which is 1024 at start-up, may be set between 10 and 1024.
<u>LOAD</u> filename	Loads <u>filename</u> into the text following the current line.
<u>LOCATE</u> string	Moves pointer forward to the first line containing <u>string</u> , which may contain leading and trailing blanks.
<u>LOCATE</u> string, *	Moves pointer forward to each occurrence of <u>string</u> between pointer's current position and the end of file.
<u>MODE</u> CKPAR	Prints characters as real characters if parity is on, as octal numbers (^nnn) if parity is off.
<u>MODE</u> COLUMN	Displays column numbers whenever INPUT mode is entered.
<u>MODE</u> COUNT start increment width	<div style="display: inline-block; vertical-align: middle;"> <math>\left\{ \begin{array}{l} \text{PRINT} \\ \text{BLANK} \\ \text{SUPPRESS} \end{array} \right\}</math> </div> <p>Turns on the automatic incremented counter.</p>
<u>MODE</u> NCKPAR	Prints all characters as if they had parity on (default).
<u>MODE</u> NCOLUMN	Turns off the column display (default).
<u>MODE</u> NCOUNT	Suspends counter incrementing (default).
<u>MODE</u> NUMBER	Displays line numbers in front of the printed line.
<u>MODE</u> NNUMBER	Turns off the line number display (default).
<u>MODE</u> NOSEMI	Turns off the line terminator function of semicolons, allowing semicolons to be used as a regular print character in INPUT mode.
<u>MODE</u> PRALL	Prints lower-case characters if the device has that capability.



<u>MODE PRUPPER</u>	Prints all characters as upper case. Precedes lower-case characters with an ^L and precedes upper-case characters with an ^U if the device is upper case only.
<u>MODE PROMPT</u>	Prints prompt characters for INPUT and EDIT modes.
<u>MODE NPROMPT</u>	Stops printing of INPUT and EDIT prompt characters (default).
<u>MODE SEMI</u>	Uses semicolons as line terminators (default).
<u>MODIFY</u> /string-2/string-1/[G] [n]	Superimposes <u>string-1</u> onto <u>string-2</u> for <u>n</u> lines. If <u>G</u> is omitted, only the first occurrence of <u>string-1</u> on each line is modified; otherwise all occurrences of <u>string-1</u> are modified.
<u>MOVE</u> buffer-1{buffer-2} /string/}	Moves <u>string</u> or contents of <u>buffer-2</u> into <u>buffer-1</u> .
<u>NEXT</u> [n]	Moves the pointer <u>n</u> lines forward or backward (default <u>n</u> = 1).
<u>NFIND</u> string	Moves the pointer down to the first line not beginning with <u>string</u> .
<u>NFIND</u> ( <u>n</u> ) string	Moves the pointer down to the first line in which <u>string</u> does not start in column <u>n</u> .
<u>NLOCATE</u> string	Finds the first line that does not contain <u>string</u> anywhere in the line.
<u>OOPS</u>	Undoes the last line changed and returns it to its status before the modification.
<u>OVERLAY</u> string	Superimposes <u>string</u> on the current line. Use tabs to start in the middle of the line. Use ! to delete existing characters. (A blank in the string leaves the old character in place.)
<u>PAUSE</u>	Returns temporarily to PRIMOS for the use of PRIMOS-level commands. START returns to the previous Editor position.
<u>POINT</u> line-number	Relocates the pointer to <u>line-number</u> .



PPRINT [first] [last]

The POSITION PRINT command prints a range of lines relative to the current position without changing the current position:

first    number of lines away  
          from current position  
          to start printing

last     relative number of lines  
          away from the current  
          position     to     stop  
          printing

If only one positive number is specified, it is interpreted as the ending-line position (last) and the default starting line is the current line.

If only one negative number is specified, it is interpreted as the beginning-line position (first) and the default ending line is the current line.

If no numbers are given the default PP -5 5, which prints from five lines above to five lines below the current position.

PRINT [n]

Prints the current line or n lines beginning with the current line. Moves the pointer to the last line printed.

PSYMBOL

Prints a list of current symbol characters and their function.

PTABSET tab-1...tab-8

Provides for a setup of tabs on devices that have physical tab stops.

PUNCH { (ASR) } [n]  
          { (PTP) }

Punches n lines on high-speed or low-speed paper-tape punch.

QUIT

Returns control to PRIMOS without filing text. If file has been modified, EDITOR warns user and asks: "OK TO QUIT?"

QF

The QUIT FINAL command lets the user QUIT out of a modified file without having the EDITOR ask if it may throw away the work file.



<u>RETYPE</u> string	The current line is replaced by <u>string</u> .																				
<u>SAVE</u> [filename]	Saves file without leaving EDITOR. If user does not specify <u>filename</u> , EDITOR saves into the file being edited and prints its name.																				
<u>SYMBOL</u> name character	Changes a symbol <u>name</u> to <u>character</u> . Current default values are:																				
<table> <tr> <th><u>Name</u></th><th><u>Default Characters</u></th></tr> <tr> <td><u>KILL</u></td><td>?</td></tr> <tr> <td><u>ERASE</u></td><td>"</td></tr> <tr> <td><u>WILD</u></td><td>!</td></tr> <tr> <td><u>BLANK</u></td><td>#</td></tr> <tr> <td><u>TAB</u></td><td>\</td></tr> <tr> <td><u>ESCAPE</u></td><td>^</td></tr> <tr> <td><u>SEMICO</u></td><td>;</td></tr> <tr> <td><u>CPROMPT</u></td><td>\$</td></tr> <tr> <td><u>DPROMPT</u></td><td>&amp;</td></tr> </table>		<u>Name</u>	<u>Default Characters</u>	<u>KILL</u>	?	<u>ERASE</u>	"	<u>WILD</u>	!	<u>BLANK</u>	#	<u>TAB</u>	\	<u>ESCAPE</u>	^	<u>SEMICO</u>	;	<u>CPROMPT</u>	\$	<u>DPROMPT</u>	&
<u>Name</u>	<u>Default Characters</u>																				
<u>KILL</u>	?																				
<u>ERASE</u>	"																				
<u>WILD</u>	!																				
<u>BLANK</u>	#																				
<u>TAB</u>	\																				
<u>ESCAPE</u>	^																				
<u>SEMICO</u>	;																				
<u>CPROMPT</u>	\$																				
<u>DPROMPT</u>	&																				
<u>TABSET</u> tab-1...tab-8	Sets up to eight logical tab stops to be invoked by the tab symbol ( ).																				
<u>TOP</u>	Moves the pointer one line before the first line of text.																				
<u>UNLOAD</u> filename [n]	Copies <u>n</u> lines into <u>filename</u> .																				
<u>UNLOAD</u> filename <u>TO</u> string	Unloads lines from current file into <u>filename</u> until <u>string</u> is found.																				
<u>VERIFY</u>	Displays each line after completion of certain commands. (Default).																				
<u>WHERE</u>	Prints the current line number.																				
<u>XEQ</u> [buffer]	Executes the contents of <u>buffer</u> . If no buffer name is given, the last command line is re-executed.																				
<u>*[n]</u>	Causes the preceding command to be repeated <u>n</u> times as in:																				

F /;D;\*10

which deletes the next ten lines that begin with / . If n is omitted, the command repeats until the bottom of the file is reached.



# F

## The Password Protection System

### INTRODUCTION

Directory passwords provide an alternative to access control lists (ACLs) as a protection system for controlling the use of files and directories. This appendix describes how to use the password system. In particular, it describes how to:

- Assign passwords to directories (PASSWD)
- Use passwords to gain access to directories
- Set specific access rights on file system objects (PROTECT)
- Convert a password directory to an ACL directory (SET\_ACCESS)
- Convert an ACL directory to a password directory (REVERT\_PASSWORD)
- Create a password subdirectory under an ACL directory (CREATE)

### Note

Since ACLs are more flexible than passwords and offer greater protection, it is expected that most users will have ACL directories. The information in this appendix is provided primarily for users of systems where passwords have been in use, and where the change to ACLs has not happened or is not yet complete.



ASSIGNING DIRECTORY PASSWORDS

You can secure your directories against unauthorized users by assigning passwords with the PASSWD command. There are two levels of passwords: owner and non-owner. If you give the owner password in an ATTACH command, you have owner status; if you give the non-owner password in an ATTACH command, you have non-owner status. If you fail to give a required password correctly in an ATTACH command, PRIMOS returns the message "Bad password" and does not execute the ATTACH command. File system objects within a password-protected directory can be given different access rights for owners and non-owners with the PROTECT command (see SETTING ACCESS RIGHTS ON FILE SYSTEM OBJECTS, below).

The PASSWD command replaces any existing password(s) on the current directory with one or two new passwords, or assigns passwords to this directory if there are none. The format is:

PASSWD [owner-password] [non-owner-password]

The owner-password is specified first; the non-owner-password follows.

For example, assume the current directory has no password:

```
OK, PASSWD US THEM
OK, PASSWD NIX TRIX
OK,
```

The first command line sets the owner password US and the non-owner password THEM on the current directory. The second command line changes the owner password from US to NIX and the non-owner password from THEM to TRIX.

Passwords may contain almost any characters, but they may not begin with a digit (0-9).

If an owner-password is specified and a non-owner-password is not specified, the default for the non-owner password is null; then, any password (except the owner password) or none allows access to this directory as a non-owner.

If the PASSWD command is given alone, without either an owner-password or a non-owner-password, then the owner-password is set to blanks and the non-owner-password is set to null. Specifying no password gives access to the directory as an owner; specifying any password gives access to the directory as a non-owner. The PASSWD command given alone is useful when you wish to remove passwords from a directory.

You must have owner status on a directory in order to use the PASSWD command.



### USING PASSWORDS TO GAIN ACCESS TO DIRECTORIES

If a directory has a password, the password must be given whenever the directory name is given. The password is entered after the name of the directory, separated by one blank space. For example, assume ROCKET is the owner password for the UFD JONES. The command:

```
ATTACH JONES ROCKET
```

connects you to JONES with owner status.

If a password is used in a pathname, apostrophes enclose the entire pathname. For example:

```
ATTACH 'MAPLE HIDDEN>BRANCH4'
```

### SETTING ACCESS RIGHTS ON FILE SYSTEM OBJECTS

Specific access rights to the objects (files, segment directories, and subdirectories) within a password-protected directory can be established by using the PROTECT command. This command sets the protection codes for users with owner and non-owner status in the directory. The format is:

```
PROTECT pathname [owner-rights] [non-owner-rights] [-REPORT]
```

<u>Variable/Option</u>	<u>Meaning</u>
<u>pathname</u>	The name of the object to be protected.
<u>owner-rights</u>	A code specifying owner's access rights to the object.
<u>non-owner-rights</u>	A code specifying the non-owner's access rights to the object.
-REPORT	Specifies that the results of executing the command be reported to the user.



The values and meanings of the protection codes are:

<u>Code</u>	<u>Rights</u>
NIL	No access of any kind allowed
R	Read only
W	Write only
D	Delete only
RW	Read and write
RD	Read and delete
WD	Write and delete
RWD	Read, write, and delete (all access)

To use the PROTECT command on objects in a directory, you must have owner status on the directory. For example:

```
PROTECT MYUFD>LETTER RWD R
```

In this example, protection codes are set on the file LETTER in the UFD MYUFD so that all access rights are given to the owner and only read access is given to the non-owner.

#### Notes

The default access codes associated with any newly created object are: RWD NIL. The owner is given all rights and the non-owner is given none. Default values for the PROTECT command, however, are: NIL NIL. Thus, the command PROTECT MYFILE denies all rights to owner and non-owner alike. The owner can always recover from this situation since he can change the protection codes again and grant himself all access rights.

Although the PROTECT command may be used to modify the protection codes of objects in ACL directories, the ACL mechanism takes precedence and the codes are ignored when the object is accessed. If the ACL directory were converted to a password directory, the protection codes would establish their designated owner and non-owner access rights. To use the PROTECT command for objects in an ACL directory, you must have protect (P) access to the directory.

#### CONVERTING A PASSWORD DIRECTORY TO AN ACL DIRECTORY

Conversion from a password directory to an ACL directory is done automatically whenever the SET\_ACCESS command (explained in Chapter 16) is given on a password directory whose parent is an ACL directory. The command will not convert a password directory whose parent is a password directory.



In order to convert a directory, you must have protect rights to the parent directory, or there must be no owner password in the directory being converted.

#### Note

To convert a directory with passwords when you do not have protect access on the parent, give the sequence of commands:

```
ATTACH directory-pathname password
PASSWD
SET ACCESS directory-pathname acl
```

If you do not have protect access on the parent directory, you must supply an acl with the SET\_ACCESS command giving yourself access rights or you will lose access to the directory after it is converted. PRIMOS will warn you if this situation is about to occur, and allow you to abort the SET\_ACCESS command.

#### Examples

1. You wish to convert the password directory HAND (in the tree GLOVE>HAND) to an ACL directory with default ACL protection. GLOVE is an ACL directory. If you are attached to GLOVE, give the command:

```
SET_ACCESS HAND
```

If you are attached elsewhere, give the command:

```
SET_ACCESS GLOVE>HAND
```

PRIMOS will respond with the "OK," prompt, and HAND will now be protected by the same ACL that protects GLOVE.

2. You wish to convert the password directory FOOT (in the tree SHOE>SOCK>FOOT) to an ACL directory with the same protection existing on SHOE. SHOE is an ACL directory, but SOCK is a password directory. The command:

```
SET_ACCESS SHOE>SOCK>FOOT
```

will fail because the parent directory of FOOT is also a password directory. To convert FOOT, you must first convert SOCK and then convert FOOT.

3. To convert an entire subtree, you may use the wildcard and treewalking capabilities of PRIMOS, explained in Chapter 18:

```
SET_ACCESS *>@@>@@ -WALK_FROM 1 -DIR
```



CONVERTING AN ACL DIRECTORY TO A PASSWORD DIRECTORY

To convert an ACL directory to a password directory, use the command:

REVERT\_PASSWORD

The command takes no arguments and converts the current directory only. You must therefore be attached to the directory you wish to convert. The following constraints also apply:

- You must have protect (P) access to the directory before you convert it.
- When you convert a directory, its original password(s) and protection codes will reactivate.
- You may not have ACL-protected subdirectories under a password directory. Therefore, if you give the REVERT\_PASSWORD command for a directory that contains ACL-protected subdirectories or any access categories, the command will fail.

Note

If you convert an ACL directory to a password directory or vice versa and wish to use the LD or LIST\_ACCESS command to check the conversion, you must reattach to the directory for the changes to take effect for you. Access rights are calculated when you first attach to the directory, and the access will not be recalculated to reflect the change until you attach somewhere else. If you convert your origin directory and subsequently use the ORIGIN command, the directory will not be converted. For the ORIGIN command to reflect the conversion, you must log out and then log in again.

Example

Consider the subtree SHIRT>SLEEVE>CUFF. You wish to convert ACL directory SLEEVE to a password directory. CUFF is also an ACL directory. If you attach to SLEEVE and give the REVERT\_PASSWORD command, the command will fail. You will get the error message at the end of the following sequence:

OK, ATTACH SHIRT>SLEEVE

OK, REVERT\_PASSWORD

Directory still contains ACL subdirectories. <Current directory>  
(revert\_password)

ER!



To convert SLEEVE to a password directory, you must first convert CUFF to a password directory. Then you can convert SLEEVE (since SLEEVE will no longer contain any ACL subdirectories). The following sequence illustrates the conversion:

```
OK, ATTACH SHIRT>SLEEVE>CUFF
OK, REVERT_PASSWORD
OK, ATTACH SHIRT>SLEEVE
OK, REVERT_PASSWORD
OK,
```

#### CREATING A PASSWORD SUBDIRECTORY UNDER AN ACL DIRECTORY

The CREATE command allows the creation of a password subdirectory under an ACL directory. You must have add (A) access to the parent directory to create a new subdirectory. The format is:

```
CREATE pathname [-PASSWORD]
```

Specifying -PASSWORD creates a password subdirectory. If the option is omitted, the CREATE command creates a subdirectory of the same type as its parent directory.

You cannot create an ACL directory under a password directory.



1940

1941

1942

1943

1944



# G

## System Information

PRIMOS supports many commands that provide useful information about the availability and current usage of system resources. The following table summarizes the information available and the commands that obtain it. The STATUS command and its options are explained more fully in the PRIMOS Commands Reference Guide. Commands concerning FTR, ACLs, Quotas, and the Spool and Batch subsystems are explained in Chapters 14, 16, 17, 4 and 11, respectively, as well as in the PRIMOS Commands Reference Guide.

### Note

Information given by any STATUS command is also given by the STATUS -ALL command.

<u>Item</u>	<u>Use</u>	<u>PRIMOS Commands</u>
Number of users	Indicates system resource usage and expected performance	STATUS USERS (user list) USERS (number of users)
User id	Identifies user id of currently used terminal	STATUS, STATUS UNITS, STATUS ME
Usenumber		STATUS ME, STATUS USERS,



<u>Item</u>	<u>Use</u>	<u>PRIMOS Commands</u>
Other usernumbers		STATUS USERS
User line number		STATUS ME, STATUS USERS
User's physical devices		STATUS ME, STATUS USERS
Open file units	Avoids conflict when using files	STATUS, STATUS UNITS
Magnetic tape units	Lists assigned units, with their logical aliases and users	STATUS DEVICE, STATUS ME
Disks in operation		STATUS, STATUS DISKS
Assigned peripheral devices	Tells what devices are available	STATUS USERS
User priorities		STATUS USERS
Your phantom user number	For logging out your phantoms	STATUS USERS, STATUS ME
Network information	Tells if network is available and what systems are configured	STATUS, STATUS NET
Current systemname		STATUS NET, STATUS UNITS
Projects with attached users	Indicates current project usage	STATUS PROJECTS
Records available	Tells how much room is available for file building, sorting, etc.	AVAIL, LIST_QUOTA
Time and date	Performs time logging in audit files	DATE
Computer time used since login	Measures program execution time	TIME
Spool queue contents	Tells if job has been printed	SPOOL -LIST
Names and status of printers	Tells if local printers are functioning	PROP -STATUS
Environment for a printer	Gives parameters for printer's operations	PROP printer-name -DISPLAY



<u>Item</u>	<u>Use</u>	<u>PRIMOS Commands</u>
Computer time used since last RDY	Measures computer use time	RDY
Batch users	Gives number of waiting, held, and executing jobs	BATCH -STATUS
Batch users	Identifies executing jobs, number of jobs per queue	BATCH -DISPLAY
Your active Batch jobs	Gives job id, status	JOB -STATUS
Your active Batch job	Gives full job parameters	JOB -DISPLAY
Batch queue status	Lists Batch queues and tells which ones are available for use	BATGEN -STATUS
Batch queue configurations	Shows environment of Batch system	BATGEN -DISPLAY
Usage and quota in records on a directory	Indicates storage space allowed and used	LIST_QUOTA [pathname]
Your access groups		LIST_GROUP
ACL protecting a file system object	Gives access information on a file system object	LIST_ACCESS [pathname]
Your remote user id's	Lists remote id's used by slaves on your behalf on remote systems	LIST_REMOTE_ID
Your active file transfers	List one-line summary on your transfers	FTR -STATUS
Your active file transfers	Gives full information on your transfers	FTR -DISPLAY



Table 1

Item	Quantity	Unit	Price	Total
1. Cement	100	kg	1.20	120.00
2. Sand	200	kg	0.80	160.00
3. Gravel	300	kg	1.50	450.00
4. Water	100	liters	0.05	5.00
5. Labor	10	hours	10.00	100.00
6. Transport	1	trip	20.00	20.00
7. Miscellaneous	1	unit	5.00	5.00
<b>Total</b>				<b>760.00</b>



# INDEX



1917



# Index

- " 2-17, 4-2
- \$\$ JOB command 11-5
- &ARGS directive (CPL) 9-7
- &CALL directive (CPL) 9-8
- &CHECK...&ROUTINE directive (CPL) 9-8
- &DATA groups, terminal input in (CPL) 9-18
- &DATA...&END directive (CPL) 9-7
- &DEBUG directive (CPL) 9-8
- &DO groups (CPL) 9-15
- &DO iteration...&END directive (CPL) 9-7
- &DO...&END directive (CPL) 9-7
- &ELSE directive (CPL) 9-14
- &EXPAND directive (CPL) 9-8
- &GOTO...&LABEL directive (CPL) 9-7
- &IF directive (CPL) 9-13
- &IF...&THEN...&ELSE directive (CPL) 9-7
- &ON...&ROUTINE directive (CPL) 9-8
- &RESULT directive (CPL) 9-8
- &RETURN directive (CPL) 9-8
- &REVERT directive (CPL) 9-9
- &ROUTINE directive (CPL) 9-8
- &SELECT directive (CPL) 9-7
- &SET\_VAR directive (CPL) 9-7
- &SEVERITY directive (CPL) 9-8
- &SIGNAL directive (CPL) 9-9
- &STOP directive (CPL) 9-8



- \* 2-12, 2-13
- ; 2-18, 4-2
- <\*> 2-13
- ? 2-18, 4-2
- ABBREV command 17-1, 17-3
- Abbreviations:
  - defining your own 17-3
  - expansion of 19-2
  - syntax suppression with 17-8
  - variables in 17-7
- Aborting Batch jobs 11-8
- Access categories 2-5, A-1, 16-7
- Access Control Lists (ACLs):
  - .ACAT suffix 16-17
  - access categories A-1, 16-7, 16-19 to 16-21
  - access rights (table) 16-3
  - add access 16-25
  - ALL access 16-26
  - appearance 16-4
  - changing access categories 16-20
  - changing access rights 16-22
  - closed systems 16-26
  - combining types of protection 16-13
  - creating access categories 16-19
  - default protection 16-6, 16-9
  - definition A-1, 16-2
  - delete access 16-25
  - deleting access categories 16-21
  - distributing rights 16-14
  - EDIT\_ACCESS 16-22
  - EDIT\_ACCESS vs. SET\_ACCESS 16-23
  - flexibility 16-1, 16-26
  - list access 16-25
  - listing access rights 16-16
  - LIST\_ACCESS command 16-16, G-3
  - NONE access 16-26
  - open systems 16-26
  - outsider rights 16-27
  - overlapping access rights 16-4
  - "owner" rights 16-27
  - partial rights 16-27
  - priority ACLs 16-24, 16-19 to 16-21, A-1
  - protect access 16-25
  - read access 16-26
  - reverting to default access 16-21
  - setting access rights 16-18, 16-20, 16-21
  - SET\_ACCESS command 16-18
  - specific ACLs 16-5
  - tips on setting access 16-25
  - types of access rights 16-2
  - types of users 16-4
  - use access 16-26
  - write access 16-26
- Access to files, controlling:
  - Access Control Lists 16-1
  - directory passwords F-1, 16-1
- Accessing remote systems 14-1, 14-5
- ACLs (See Access Control Lists)
- Addressing modes 5-5, 5-6
- Advanced Text Management 1-11
- Aliases (logical) for magnetic tape drives 13-7, 13-9
- APPEND, Editor command 4-12
- APPLIB (R-mode library) 15-3
- Application subroutines 15-1
- Applications libraries 15-3
- Argument A-2
- Arguments (CPL):
  - &ARGS directive 9-7
  - multiple arguments 9-10
  - omitted arguments 9-11, 9-12
- ASCII Character Set:
  - non-printing C-2
  - printing C-4



- ASCII files 12-1
- Assembly Language Programmer's Companion 1-12
- Assembly Language Programmer's Guide 1-8
- ASSIGN command:
  - card reader 13-2
  - magnetic tape drives 13-2, 13-4 to 13-12
  - messages 13-11
  - paper tape reader 13-2, 13-4
- Assigning magnetic tape drives 13-2, 13-4 to 13-12
- Assigning peripheral devices 13-2, 13-4 to 13-12
- ATTACH command 3-6
- Attach point, initial A-8
- Attaching, across a network 14-1, 14-3
- ATTN key 2-17
- AVAIL command G-2
- Backslash (\) 2-17, 4-3
- BASIC 1-8
- BASIC/VM Programmer's Companion 1-12
- BASIC/VM Programmer's Guide 1-8
- BATCH command G-3, 11-10
- Batch jobs:
  - aborting 11-8
  - cancelling 11-7
  - displaying Batch information 11-8
  - executing 11-1 to 11-12
  - execution environment 7-2
  - modifying 11-6
  - monitoring 11-8
  - restarting 11-7
  - sending messages from 11-10
  - submitting 11-2
- Batch queues 11-11
- BATGEN command G-3, 11-12
- Binary code 2-3
- Binary files 2-3, 5-4, A-2
- Binary search 15-12
- BOTTOM, Editor command 4-8
- BREAK key 2-17
- BREAKPOINT, DBG subcommand 8-5
- Byte A-2
- Cancelling a spool request 4-25
- Cancelling Batch jobs 11-7
- Cards:
  - control 13-3
  - reading 13-2
  - using 13-1
- Caret (^) 2-17
- Category name A-2
- Chaining command files 10-4
- Change bars 1-1
- CHANGE, Editor command 4-12
- Changes in this edition 1-1
- CHANGE\_PASSWORD command 3-7
- Changing directories 3-6
- Changing login passwords 3-7
- Changing the system prompts 17-1, 17-2
- Characters:
  - control 2-16, 2-17
  - reserved 2-18
  - special 2-16, 2-17



CLEAR, DBG subcommand 8-7

Closing command files:

input 10-6  
output 10-7

CMPF command 12-7

CNAM\$\$ (sample subroutine)  
15-16

CNAME command 3-11

COBOL Programmer's Companion  
1-12

COBOL Reference Guide 1-5

COBOL:

command 5-3  
compiler 5-3  
compiler defaults 5-4  
documentation 1-5, 1-12  
mode generated 5-6  
used with other languages 5-7

Code generation 5-5

Column display 4-4

Combining languages in a program  
5-7

Combining program modules 5-7

COMINPUT:

command 10-2  
login file 17-13  
options 10-2

Command environment 17-1

Command files:

(See also CPL)  
chaining 10-4  
closing 10-6  
definition 10-1  
errors (COMINPUT) 10-5  
input 10-2  
output 10-6  
PHANTOM 10-10

Command line features:

abbreviation expansion 19-2  
combining 18-22  
iteration 18-1 to 18-3, 19-5  
multiple command processing  
19-4  
name generation 18-1, 18-19  
to 18-22, 19-7  
syntax suppression 19-3  
treewalking 18-1, 18-11 to  
18-18, 19-5  
variable and function  
evaluation 19-4  
wildcards 18-1, 18-4 to  
18-11, 19-6

Command line processing:

discussion 19-1 to 19-18  
example 19-7 to 19-18

Command output files 10-6

Command Procedure Language (See  
CPL)

Command processing, controlling  
18-1

Commands:

\$\$ JOB 11-5  
ABBREV 17-1, 17-3  
abbreviating 17-3  
ASSIGN 13-2  
ASSIGN: 13-5  
ATTACH 3-6  
AVAIL G-2  
BATCH G-3, 11-10  
BATGEN G-3, 11-12  
CHANGE\_PASSWORD 3-7  
CMPF 12-7  
CNAME 3-11  
COBOL 5-3  
COMINPUT 10-2  
COMOUTPUT 3-18, 10-6  
CONCAT 12-12  
COPY 3-11  
CPL 9-2, 9-4  
CREATE 3-8  
CRMPC 13-3  
DATE G-2, 10-8  
DBG 8-4, 20-3  
DEFINE\_GVAR 17-9  
DELETE 3-14  
DELETE\_VAR 17-9



DMSTK 20-3  
 ED 4-1  
 EDIT\_ACCESS 16-22  
 essential (table) 3-3  
 F77 5-3  
 FILMEM 6-7  
 FTN 5-3  
 FTR G-3, 14-9  
 HELP 3-17  
 JOB 9-4, G-3, 11-2  
 LD 3-9, 17-23  
 LIST\_ACCESS G-3, 16-16  
 LIST\_GROUP G-3  
 LIST\_PRIORITY\_ACCESS 16-24  
 LIST\_QUOTA G-2, G-3, 17-21  
 LIST\_REMOTE\_ID G-3  
 LIST\_VAR 17-9  
 LOAD 6-1, 6-6 to 6-9  
 LOGIN 3-2 to 3-5  
 LOGOUT 3-20  
 MAGRST 13-19  
 MAGSAV 13-15  
 MESSAGE 2-19, 2-20, 11-10,  
 17-1, 17-14  
 MRGF 12-9  
 NETLINK 14-5  
 ORIGIN 3-7  
 PASCAL 5-3  
 PHANTOM 9-4, 10-10  
 PLIG 5-3  
 PROP G-2  
 PROTECT F-3, F-4  
 RDY G-3, 10-9, 17-1  
 RESUME 6-7, 9-2, 9-4  
 RLS 20-3  
 RPG 5-3  
 SEG 6-1, 6-2 to 6-6, A-10  
 SET\_ACCESS 16-18  
 SET\_DELETE 3-15  
 SET\_QUOTA 17-19  
 SET\_VAR 17-9  
 SIZE 17-23  
 SLIST 3-9  
 SORT 12-1, 12-2  
 SPOOL 4-24 to 4-27, G-2  
 START 7-4, 20-2  
 STATUS G-1, G-2, 13-8, 14-2,  
 14-4  
 stopping 3-19  
 TERM 2-19  
 TIME G-2, 10-8  
 UNASSIGN 13-2, 13-12  
 VRPG 5-3

## Comments:

in command files 10-2  
 in source files 4-4

Communications, documentation  
 for 1-10

## COMOUTPUT:

command 3-18, 10-6  
 files 9-4  
 options 10-7

Comparing files 12-7

Compatibility 2-2

## Compilers:

COBOL 5-3  
 defaults 5-4  
 F77 5-3  
 FTN 5-3  
 messages 5-7  
 PASCAL 5-3  
 PLIG 5-3  
 RPG 5-3  
 VRPG 5-3

Compiling programs 5-1 to 5-7

Component A-2

Compressed files 12-1

CONCAT command 12-12

## Condition mechanism:

definition A-3, 20-1  
 on-units 15-18, 20-2 to 20-5  
 subroutines 15-18

## Constants:

execution B-4  
 segmented loader (SEG) B-4  
 virtual loader (LOAD) B-4

CONTINUE, DBG subcommand 8-4

CONTROL key 2-16

CONTROL-P 2-18, 3-19

CONTROL-Q 2-18



CONTROL-S 2-18

Controlling access to files:  
 Access Control Lists 16-1  
 directory passwords F-1, 16-1

Controlling command processing  
 18-1

Conversion subroutines 15-4

COPY command 3-11

Copy file system objects 3-11  
 to 3-14

Correspondence management 1-11

CPL (Command Procedure  
 Language):  
 .CPL suffix 9-2, 9-4  
 arguments used in 9-10 to  
 9-12  
 branching in 9-12  
 command 9-2  
 creating programs in 9-2  
 debugging 9-4  
 directives 9-7 to 9-9, 9-13,  
 9-14  
 ending programs 9-20  
 executing directives in 9-3  
 executing programs in 9-2  
 features 9-1  
 functions used in 9-16  
 interpreter 9-2  
 logic errors in 9-5  
 login files 17-10  
 messages from Batch jobs  
 11-10  
 null strings in 9-11  
 PRIMOS commands in 9-5  
 subsystems used with 9-17  
 syntax errors in 9-4  
 variables used in 9-9

CPL User's Guide 1-7

CPU A-3

CREATE command 3-8

Creating "listing files" 5-5

CRMPC command 13-3

Cross-reference listings  
 (created by compiler) 5-5

Current directory 2-12, A-3

Current disk 2-13

Data base management,  
 documentation for 1-10

DATE command G-2, 10-8

DBG command 8-4, 20-3

DBG (See Debugger, Source-Level)

DBMS Administrator's Guide 1-10

DBMS COBOL Reference Guide 1-10

DBMS FORTRAN Reference Guide  
 1-10

DBMS Schema Reference Guide  
 1-10

DBMS/QUERY Reference Guides  
 1-10

DBMS/QUERY Report Generator  
 Casebook 1-10

DBMS/QUERY User's Guide 1-10

DC, LOAD subcommand 6-8

Debugger, Source-Level (DBG):  
 ":" (evaluate symbol) 8-7  
 action list on breakpoint 8-1  
 assignment 8-3  
 BREAKPOINT 8-5  
 breakpointing 8-1, 8-5  
 built-in (intrinsic) functions  
 8-3  
 calling subroutine or function  
 8-2  
 CLEAR 8-7  
 clearing breakpoints 8-7  
 compiling with -DEBUG 8-4  
 conditional breakpointing 8-2  
 CONTINUE subcommand 8-4  
 entry/exit tracing 8-2



- examine expression type 8-3
  - examining and modifying data 8-7
  - examining the source code 8-8
  - expression evaluation 8-3
  - invoking 8-4
  - LET 8-7
  - LIST 8-7
  - LISTALL 8-7
  - listing breakpoints 8-7
  - program restart 8-2
  - RESTART subcommand 8-4
  - sample debugging session 8-11 to 8-14
  - single stepping 8-1, 8-5
  - SOURCE 8-5, 8-8
  - source file examination 8-3
  - SOURCE subcommand arguments 8-9
  - starting program execution 8-4
  - statement tracing 8-2
  - STEP 8-5
  - stopping program execution 8-5
  - terminating 8-4
  - traceback 8-2
  - tracepointing 8-2
  - transfer of control 8-2
  - value of variable 8-3
  - value tracing 8-2
- Decision-making, in CPL programs 9-12
- Default protection (ACLs) 16-6, 16-9
- Defaults:
- concept of 2-14
  - Editor (ED) B-2
  - PRIMOS keyboard B-2
  - protection B-5
  - terminal B-1
- DEFINE\_GVAR command 17-9
- DELETE/DEL key 2-17
- DELETE:
- Editor command 4-13
  - PRIMOS command 3-14
  - SEG subcommand 6-3
- DELETE\_VAR command 17-9
- Deleting file system objects 3-14
- Devices:
- assigning 13-2
  - definition A-3
  - releasing 13-2, 13-12
- Directives, CPL, summary of 9-7 to 9-9
- Directories:
- changing 3-6
  - creating 3-8
  - current 2-12
  - definition 2-4, A-3
  - listing contents of 3-9
  - MFDs 2-4
  - origin 2-12, 3-5, A-8
  - password-protected 2-13, F-3, F-4, F-6, F-7
  - segment 6-1
  - sub-UFDs 2-4
  - UFDs 2-4
- Directory name A-3
- Directory passwords (See Passwords, directory)
- Disconnected from systemname 14-2
- Disk:
- quotas (See Quotas, disk)
  - sorts 15-11
  - volume 2-9, 2-11
- Displaying Batch information 11-8
- Displaying tape drive information 13-8
- Distributed Processing Terminal Executive Guide 1-11
- DMSTK command 20-3
- DPTX (Distributed Processing Terminal Executive) 1-11



DUNLOAD, Editor command 4-4,  
4-16

ED (line-oriented editor):  
(See also EDITOR commands)  
command 4-1  
defaults B-2  
EDIT mode 4-2  
Editor symbols B-3  
general 1-9  
INPUT mode 4-2  
special characters 4-3, 4-4  
useful techniques 4-3 to 4-5

EDIT Mode 4-2

#### EDITOR commands:

APPEND 4-12  
BOTTOM 4-8  
CHANGE 4-12  
DELETE 4-13  
DUNLOAD 4-4, 4-16  
FILE 4-3, 4-19  
FIND 4-4, 4-10  
IB 4-14  
INSERT 4-14  
LOAD 4-17  
LOCATE 4-9  
MODE COLUMN 4-4  
MODE NOSEMI 4-5  
MODE SEMI 4-5  
MODIFY 4-4  
NEXT 4-8  
NFIN 4-10  
OOPS 4-15  
OVERLAY 4-4  
POINT 4-9  
PRINT 4-7  
QUIT 4-19  
RETYPE 4-14, 4-15  
SAVE 4-3, 4-20  
summary of E-1 to E-6  
SYMBOL 4-5  
TABSET 4-4  
TOP 4-8  
UNLOAD 4-18

Editor defaults (ED) B-2

EDIT\_ACCESS command 16-22

Electronic mail 1-11

EMACS (screen-oriented editor)  
1-9

EMACS Extension Writing Guide  
1-9

EMACS Primer 1-9

EMACS Reference Guide 1-9

Entryname A-6

Error messages:  
ASSIGN 13-11  
ATTACH 3-6  
Batch D-32 to D-41  
condition mechanism D-41 to  
D-53  
LOAD loader D-7 to D-9  
MESSAGE 17-17  
run-time D-10 to D-31  
SEG loader D-2 to D-9

Error-handling, system (See  
Condition mechanism)

Errors:  
in command files 10-5  
runtime 7-5

Execution:  
constants B-4  
PHANTOM files 10-11  
R-mode runfiles 7-4  
segmented runfiles 7-3

External command A-3

F77 command 5-3

FED (Forms Editor) 1-10

FED User's Guide 1-10

File protection keys A-5

File system objects:  
changing names of 3-11  
copying 3-11 to 3-14  
definition A-4  
deleting 3-14  
naming rules 2-6, 4-19  
pathnames of 2-9  
protecting 3-15, 3-17



- storage of 2-8
  - suffixes 2-6, 2-7
  - types of 2-3
- File system:
- structure 2-3 to 2-13
  - subroutines 15-3
  - using the 2-3 to 2-13
- File Transfer Service (FTS):
- access rights needed 14-10
  - cancelling requests 14-7
  - checking requests 14-13
  - fetching a file 14-12
  - FTR's help facility 14-18
  - introduction 14-1, 14-9
  - logging request events 14-15
  - printing at remote sites 14-12
  - request names and request numbers 14-11
  - requesting notification 14-16
  - requests on hold 14-18
  - sending a file 14-11
  - server 14-10
  - source and destination sites 14-11
- FILE, Editor command 4-3, 4-19
- File-unit A-4
- Filenames 4-19, A-4
- Files:
- ASCII 12-1
  - binary 5-4
  - COMINPUT 10-2
  - command 10-1
  - COMOUTPUT 9-4, 10-6, 17-13
  - comparing 12-7
  - compressed 12-1
  - concatenating 12-12
  - creating 4-1
  - definition 2-3, A-4
  - examining contents 3-9
  - fixed-length 12-1
  - joining sequentially 12-12
  - merging (MRGF) 12-6, 12-9
  - object 5-4
  - on cards 13-1
  - on magnetic tape 13-1
  - on paper tape 13-1
  - output 10-6
  - phantom 10-10
  - sorting 12-1 to 12-7
  - types 12-1
  - variable-length 12-1
  - with CMPF 12-7
- FILMEM command 6-7
- FIND, Editor command 4-4, 4-10
- Fixed-length files 12-1
- FORMS 1-10
- FORMS Programmer's Guide 1-10
- FORTRAN 77 Programmer's Companion 1-12
- FORTRAN 77 Reference Guide 1-5
- FORTRAN 77:
- combined with other languages 5-7
  - compiler 5-3
  - compiler defaults 5-4
  - documentation 1-5, 1-12
  - modes generated 5-6
  - on-units in 20-2
  - subroutine keys and error codes 15-2
- FORTRAN IV:
- compiler 5-3
  - on-units in 20-2
  - subroutine keys and error codes 15-2
- FORTRAN Programmer's Companion 1-12
- FORTRAN Reference Guide 1-5
- FORTRAN:
- combined with other languages 5-7
  - compiler defaults 5-4
  - documentation 1-5, 1-12
  - editing source files 4-4, 4-7
  - example with DBG 8-11 to 8-14
  - modes generated 5-6
  - sample program with DBG 8-6



- FIN command 5-3
- FTR command G-3, 14-9
- FTS (See File Transfer Service)
- Functions:
  - CPL 9-16
  - general 17-8
- Global variables:
  - defining 17-1
  - using 17-9
- Hard copy, printing 4-25
- Hardware features 2-2
- HELP command 3-17
- HELP, SEG subcommand 6-3
- High-level languages:
  - compiling 5-1 to 5-7
  - loading 6-1
- I-mode 5-6, 6-1
- IB, Editor command 4-14
- Identifier A-4
- Identity A-4
- In-memory sorts 15-12
- Initial attach point A-8
- INITIALIZE:
  - LOAD subcommand 6-8
  - SEG subcommand 6-3
- INPUT Mode 4-2
- INSERT, Editor command 4-14
- Interactive execution 7-2, 7-3
- Internal command A-5
- Interpretive BASIC 1-8
- Interpretive BASIC Programmer's Guide 1-8
- INTRPT key 2-17
- Iteration lists 18-2
- Iteration:
  - creating iteration lists 18-2
  - cross-product 18-3
  - definition 18-1, 19-5
  - lists as parts of arguments 18-3
  - multiple iteration lists 18-2
- JOB:
  - command 9-4, G-3, 11-2
  - command options 11-3
- Joining files sequentially 12-12
- Keyboard, terminal 2-15
- Keys, file protection A-5
- Keys, for sorts 12-3
- Languages:
  - BASIC 1-8
  - COBOL 1-5, 5-3
  - CPL (Command Procedure Language) 9-1
  - FORTRAN 77 1-5, 5-3
  - FORTRAN 1-5, 5-3
  - Pascal 1-5, 5-3
  - PL/I 1-7
  - RPG 1-7, 5-3
- LD command 3-9, 17-23
- Ldev A-5
- Ldisk A-5
- Ldn (logical device number) 13-5, 13-6
- LET, DBG subcommand 8-7
- Libraries, subroutine:
  - applications 15-3
  - general 15-1
  - sort and search 15-10



- system 15-13
- LIBRARY:
  - LOAD subcommand 6-8
  - SEG subcommand 6-3
- LIST, DBG subcommand 8-7
- LISTALL, DBG subcommand 8-7
- Listing files (created by compilers) 5-5
- LIST\_ACCESS command G-3, 16-16
- LIST\_GROUP command G-3
- LIST\_PRIORITY\_ACCESS command 16-24
- LIST\_QUOTA command G-2, G-3, 17-21
- LIST\_REMOTE\_ID command G-3
- LIST\_VAR command 17-9
- LOAD:
  - command and subcommands 6-1, 6-6 to 6-9
  - constants B-4
  - Editor command 4-17
  - LOAD subcommand 6-8
  - SEG subcommand 6-3
- Loading and Debugging Programmer's Companion 1-12
- Loading procedures:
  - with LOAD 6-7
  - with SEG 6-3
- Loading programs 6-1 to 6-9
- LOCATE, Editor command 4-9
- Locating text lines 4-4
- Logging in 3-2 to 3-5
- Logging out 3-20
- Logical device number (ldn) 13-5, 13-6
- Logical disk numbers 2-11
- Logical disk:
  - definition A-5
  - names 14-3, 14-4
  - numbers 2-11, A-5, 14-4
- Login (command) files:
  - COMINPUT 17-13
  - CPL 17-10
  - definition 17-1
  - examples 17-12, 17-14
  - R-mode runfiles 17-14
- LOGIN command 3-2 to 3-5
- Login, across network 14-1, 14-2
- LOGOUT command 3-20
- Logout notification, PHANTOM 10-12
- Logout, PHANTOM 10-12
- MAGNET utility 13-14
- Magnetic tapes:
  - assigning tape drives 13-2, 13-4 to 13-12
  - mounting 13-10
  - operator assistance 13-10
  - using 13-1, 13-3 to 13-24
- MAGRST:
  - command 13-19
  - dialog 13-19
  - protection 13-23
- MAGSAV:
  - command 13-15
  - dialog 13-16
  - options 13-15
  - protection 13-18
- MAP:
  - LOAD subcommand 6-8
  - SEG subcommand 6-3



- Master File Directories (MFDs)
  - 2-4, A-5
- Mathematical subroutines 15-4
- Merge sorts 12-2
- Merging files 12-2, 12-6, 12-9
- Merging sorted files 12-2
- MESSAGE command 2-20, 11-10, 12-19, 17-1, 17-14
- Messages:
  - (See also Error messages)
  - compiler 5-7
  - user-to-user 17-14
- MFDs 2-4, A-5
- MIDAS (Multiple Index Data Access System) 1-9
- MIDAS User's Guide 1-9
- Mode A-6
- MODE COLUMN, Editor command 4-4
- MODE NOSEMI, Editor command 4-5
- MODE SEMI, Editor command 4-5
- MODE, LOAD subcommand 6-8
- MODIFY, Editor command 4-4
- Modifying Batch jobs 11-6
- Modifying text lines 4-4
- Monitoring Batch jobs 11-8
- Monitoring Batch queues 11-11
- Monitoring speed of execution 10-9
- Monitoring tape drive usage 13-8
- Mounting magnetic tapes 13-10
- Moving lines of code 4-4
- MRGF command 12-9
- Multiple command processing 19-4
- Name generation:
  - adding components 18-21
  - definition 18-1, 18-19, 19-7
  - deleting components 18-22
  - examples 18-21
  - generation patterns (table) 18-20
  - source pathname 18-19
- Naming file system objects (rules) 2-6, 4-19
- Nested IF directive (CPL) 9-14
- NETLINK command:
  - basic usage 14-6
  - HELP facility 14-9
  - MODE REMOTE ECHO option 14-8
  - overview 14-5
  - TO option 14-7
- Networks:
  - accessing remote 14-5
  - attaching across 14-3
  - defined 14-1
  - disk names 14-3
  - logging in across 14-1, 14-2
  - STATUS 14-2
  - UFDs with the same name 14-4
  - using 14-1 to 14-20
- New User's Guide to Editor and Runoff 1-9
- NEXT, Editor command 4-8
- NFIND, Editor command 4-10
- Nodename A-6
- Nontag sorts 12-3
- Number representations A-6



- OAS (Office Automation Systems) 1-11
- OAS Advanced Text Management Guide (PT65) 1-11
- OAS Management Communications and Support Guide (PT45) 1-11
- OAS Management Communications and Support Guide (PT65) 1-11
- OAS System Administrator's Guide 1-11
- OAS Word Processing Guide (PT45) 1-11
- OAS Word Processing Guide (PT65) 1-11
- Object code 2-3
- Object files 2-3, 5-4, A-6
- Objectnames 2-6, 2-12, A-6, A-7
- Objects, file system (See File system objects)
- Office Automation, documentation for 1-11
- On-units:
  - actions 20-3
  - definition A-8
  - FORTTRAN example 20-6
  - scope 20-5
  - subroutines 15-18
  - system 20-2, 20-3
  - user-written 20-4
  - writing 20-4
- OOPS, Editor command 4-15
- Open state, for file units A-8
- Operating system 2-2
- Operator intervention (with magnetic tape) 13-10
- Option A-8
- Order of loading:
  - with LOAD 6-9
  - with SEG 6-5
- Ordinary pathname 2-11
- ORIGIN command 3-7
- Origin directory 2-12, 3-5, A-8
- Output stream A-9
- Output, written to a file 10-6
- OVERLAY, Editor command 4-4
- Overlaying comments 4-4
- Packname A-9
- Page A-9
- Parsing subroutines 15-4
- Partition A-9
- PASCAL command 5-3
- Pascal Reference Guide 1-5
- Pascal:
  - compiler 5-3
  - compiler defaults 5-4
  - documentation 1-5
  - modes generated 5-6
  - subroutine keys and error codes 15-2
- Password-protected directories:
  - converting from ACLs to passwords F-6
  - converting from passwords to ACLs F-4
  - creating under ACL directory F-7
  - non-owner rights, setting F-3
  - owner rights, setting F-3
  - pathnames with 2-13
  - PROTECT command F-4
  - protection codes (for files) F-4



- Passwords, directory:  
     alternative to ACLs F-1, 16-1  
     assigning F-2  
     general 2-9, 2-13  
     using with commands F-3
- Passwords, login 3-4, 3-7
- Pathnames 2-9, 2-11, 2-12, A-9
- Pathnames, for access to files  
     on remote disks 14-1
- Pdev A-9
- Pdisk A-9
- Pdn (physical device number)  
     13-5, 13-6
- Peripheral devices:  
     assigning 13-2, 13-4 to 13-12  
     releasing 13-2, 13-12
- PHANTOM command 9-4, 10-10
- Phantoms:  
     as network slaves 14-19  
     execution environment 7-2  
     phantom user A-9  
     using 10-10 to 10-15
- Physical device number (pdn)  
     13-5
- PL/I Subset G Reference Guide  
     1-7
- PL/I Subset G:  
     compiler 5-3  
     documentation 1-7  
     editing source files 4-4  
     modes generated 5-6  
     on-units in 20-2  
     subroutine keys and error codes  
         15-2  
     used with other languages 5-7
- PLLG:  
     command 5-3  
     compiler defaults 5-4
- PMA:  
     on-units in 20-2  
     subroutine keys and error codes  
         15-2
- POINT, Editor command 4-9
- POWER 1-9
- PRIME/POWER Companion 1-12
- PRIME/POWER Guide 1-9
- PRIMENET 1-10, 14-1 to 14-20
- PRIMENET Guide 1-10
- PRIMOS Commands Programmer's  
     Companion 1-12
- PRIMOS Commands Reference Guide  
     1-7
- PRIMOS:  
     command line standards B-2  
     keyboard defaults B-2  
     keyboard standards B-1  
     Prime's operating system 2-2
- PRINT, Editor command 4-7
- Printing files 4-24 to 4-27
- Priority ACLs:  
     definition 16-24, A-1  
     listing 16-24
- Procedure A-10
- Process A-10
- Program environments 7-1
- Program with DBG 8-6
- Programmer's Companions 1-12
- Project id 3-4
- Projects 3-4
- Prompts:  
     changing 2-14, 17-1, 17-2  
     ER! 2-14



- OK, 2-14
- PROP command G-2
- PROTECT command F-3, F-4
- Protecting files and directories 16-1
- Protection defaults B-5
- PSD (Prime Symbolic Debugger) 8-3
- Public Data Networks 14-5
- Purging file system objects 3-14
- Question mark (?) 2-18
- Queues, BATCH 11-11
- QUIT:
  - Editor command 4-18
  - LOAD subcommand 6-8
  - SEG subcommand 6-3
- Quotas, disk:
  - allocating storage space 17-18
  - below current usage 17-27
  - calculating storage availability 17-25
  - definition A-10, 17-18
  - "disk-full" condition 17-27
  - listing with LD 17-22
  - listing with LQ 17-21
  - modifying 17-20
  - recovering from overloads 17-30
  - removing 17-21
  - setting 17-19
  - setting useful 17-28
  - tips on using 17-27
  - using LD -SIZE 17-23
  - using SIZE 17-23
- Quotation marks ("), double 2-17
- R-mode 5-5, 6-1, A-10
- R-mode runfile, executing 7-4
- RDY command G-3, 10-9, 17-1
- Reading punched cards 13-2
- Reading punched paper tape 13-4
- Recording terminal sessions 3-17
- Relative Pathnames 2-12
- Relinquishing peripheral devices 13-2, 13-12
- Remote ids:
  - defined 14-19
  - establishing 14-19
  - examining 14-20
- Remote login 14-1, 14-2
- Reserved characters 2-18, A-10
- Resolving discrepancies in files:
  - with CMPF 12-7
  - with MRGF 12-9
- RESTART, DBG subcommand 8-4
- Restarting Batch jobs 11-7
- Restoring files from tape to disk (MAGRST) 13-19
- RESUME command 6-7, 9-2, 9-4
- RETURN key 2-17
- RETURN, SEG subcommand 6-3
- RETYPE, Editor command 4-14, 4-15
- Ring protection system 2-2
- RLS command 20-3
- RPG II Debugging Template 1-7



RPG II Reference Guide 1-7

## RPG II:

compiler 5-3  
V-mode compiler 5-3

## RPG:

command 5-3  
compiler defaults 5-4  
documentation 1-7  
mode generated 5-6

RUBOUT key 2-17

Runfile A-10

Running jobs under Batch 11-1  
to 11-12

Running R-mode programs 7-4

Running segmented runfiles 7-3

RUNOFF (text formatter) 1-9

Runtime error messages 7-5

## SAVE:

Editor command 4-3, 4-20  
LOAD subcommand 6-8  
SEG subcommand 6-3

Saving disk files on tape  
(MAGSAV) 13-15Security (on file system  
objects):

Access Control Lists (ACLs)  
16-1 to 16-27  
directory passwords 16-1,  
F-1 to F-7

SEG (segmented loader):  
command and subcommands 6-1,  
6-2 to 6-6  
constants B-4  
definition A-10

Segment A-11

## Segment directories:

creating 6-1  
definition 2-4, A-11  
vs. "directories" 2-5

Segmentation 2-2

Segno A-11

Semicolon (;) 2-18

Sending messages from Batch jobs  
11-10Server, File Transfer Service  
(FTS) 14-10Setting terminal characteristics  
2-19

SET\_ACCESS command 16-18

SET\_DELETE command 3-15

SET\_QUOTA command 17-19

SET\_VAR command 17-9

SIZE command 17-23

Slaves (phantoms) 14-19

SLIST command 3-9

Sort and search libraries 15-10

SORT command 12-1, 12-2

Sorting files 12-1 to 12-7

Source file A-11

SOURCE, DBG subcommand 8-5,  
8-8Source-Level Debugger (See  
Debugger, Source-Level)

Special characters 2-17

Special characters, Editor 4-2

Special terminal keys 2-16

Specific ACLs 16-5

SPOOL command 4-24 to 4-27,  
G-2



- Spooling files:
  - cancelling a request 4-25
  - changing the header 4-26
  - deferring printing 4-26
  - eliminating headers 4-27
  - listing the queue 4-25
  - multiple options 4-27
  - printing at specific locations 4-27
  - printing multiple copies 4-25
  - printing on special forms 4-26
  - requesting hard copy 4-24
- SRTLIB (R-mode sort library) 15-12
- Standards:
  - PRIMOS command line B-2
  - PRIMOS keyboard B-1
- START command 7-4, 20-2
- STATUS command:
  - alone G-1
  - DEVICE G-2, 13-8
  - DISKS G-2, 14-4
  - ME G-1
  - NETWORK G-2, 14-2, 14-3
  - phantom information 10-13
  - PROJECTS G-2
  - UNITS G-1
  - USERS G-1
- STEP, DBG subcommand 8-5
- Stopping unwanted commands 3-19
- String manipulation subroutines 15-3
- Sub-UFDs 2-4, A-11
- Subdirectories 2-4, A-11
- Subroutines Reference Guide 1-7
- Subroutines:
  - APPLIB (R-mode library) 15-3
  - application 15-1
  - applications libraries 15-3
  - binary search 15-12
  - CNAM\$\$ (sample subroutine) 15-16
  - condition mechanism 15-18
  - conversion 15-4
  - description (CNAM\$\$) 15-16
  - descriptions (system routines) 15-13
  - descriptions, applications libraries 15-4
  - file system 15-3
  - for condition mechanism interface 20-4
  - in-memory sorts 15-12
  - keys and error codes 15-2
  - libraries 15-1
  - mathematical 15-4
  - parsing 15-4
  - sort and search libraries 15-10
  - sort characteristics (table) 15-12
  - SRTLIB sort library 15-12
  - string manipulation 15-3
  - system information 15-4
  - system libraries 15-1, 15-13
  - used in sample program 15-17
  - user query 15-4
  - VAPPLIB (V-mode library) 15-3
  - VSRTL I sort library 15-11
- Subsystems, used with CPL &DATA groups (CPL) 9-17
- Suffixes:
  - compiler 5-1, 5-2
  - conventions 2-6, 2-7, A-7
  - language 5-2
  - objectname A-11
  - recognized 2-7, 5-2
  - recommended 2-7
- SYMBOL, Editor command 4-5
- Syntax suppression 19-3
- System Administrator Programmer's Companion 1-12
- System Administrator's Guide 1-7
- System Architecture Reference Guide 1-8



System Operator's Guide 1-7

System:

defaults and constants B-1  
 information (table) G-1 to  
 G-3  
 PRIMOS keyboard standards B-1  
 prompts 2-14  
 subroutines 15-13  
 system information subroutines  
 15-4  
 terminal defaults B-1

Systemname A-11

Tab settings 4-3

TABSET, Editor command 4-4

Tagsorts 12-3

Tape drives:

assigned by operator 13-4,  
 13-5, 13-10  
 assigned by user 13-4, 13-5  
 assignment forbidden 13-5  
 logical aliases for 13-5,  
 13-7, 13-9  
 logical device number (ldn)  
 13-5, 13-6  
 monitoring usage 13-8  
 physical device number (pdn)  
 13-5, 13-6

Terminal controls and switches  
 2-16

Terminal defaults B-1

Terminal keyboard 2-15

Terminal keys, special 2-16

Text editing:

documentation for 1-9  
 ED 1-9  
 EMACS 1-9  
 RUNOFF 1-9

TIME command G-2, 10-8

TOP, Editor command 4-8

Transferring files between  
 systems (See File Transfer  
 Service)

Tree structure 2-8

Treenames 2-9, A-11

Treewalking:

definition 18-1, 18-11, 19-5  
 examples 18-17, 18-18  
 options (table) 18-16  
 processing with 18-14  
 sample terminal display 18-13

Type-ahead 2-14

UFDs:

definition 2-4, A-11  
 with the same name (networks)  
 14-4

UNASSIGN command 13-2, 13-12

Underscore ( ) 2-18

Unit A-12

UNLOAD, Editor command 4-18

User File Directories (UFDs)  
 2-4

User ids:

definition 3-2  
 obtaining 3-2

User query subroutines 15-4

Users, maximum number of 2-2

V-mode:

general 5-5, A-12  
 loading 6-1

VAPPLB (v-mode library) 15-3

Variable-length files 12-1

Variables:

at command level 17-9  
 defining global 17-1  
 evaluation with functions  
 19-4



- in abbreviations 17-7
- Virtual loader (LOAD) constants  
B-4
- Virtual memory 2-2
- Volume 2-9, A-12
- Volume-name A-12
- VPSP (Virtual Prime Symbolic  
Debugger) 8-3
- VRPG:
  - command 5-3
  - compiler defaults 5-4
  - mode generated 5-6
- VSRTLI (V-mode sort library)  
15-11
- Wait 14-2
- Wildcards:
  - date-selection options 18-8
  - definition 18-1, 18-4, 19-6
  - examples 18-5
  - inverted matching 18-6
  - options (table) 18-7
  - type-designation options 18-8
  - verification options 18-10
  - wild characters (table) 18-4
  - wildcard matching 18-5
- Word A-12
- Writing output to a file 10-6
- XOFF 2-19
- \ 2-17, 4-3
- ^ 2-17, 4-2
- \_ 2-18



1. The first part of the paper  
deals with the general  
principles of the  
theory of the  
relativity of  
simultaneity.  
It is shown that  
the laws of physics  
are the same in  
all inertial frames  
of reference.

2. The second part of the paper  
deals with the  
consequences of the  
theory of the  
relativity of  
simultaneity.  
It is shown that  
the laws of physics  
are the same in  
all inertial frames  
of reference.

3. The third part of the paper  
deals with the  
consequences of the  
theory of the  
relativity of  
simultaneity.  
It is shown that  
the laws of physics  
are the same in  
all inertial frames  
of reference.



## USER SURVEY

Tell us how we're doing, and we'll send you a free Programmer's Companion.

Your name \_\_\_\_\_

Company or School \_\_\_\_\_

Address \_\_\_\_\_

City, State, Zip \_\_\_\_\_

1. What is your job title or function? \_\_\_\_\_

2. What specific task describes what you do? \_\_\_\_\_

3. Does your company or school own a Prime computer?

☐ YES ☐ NO

a. If YES, which model?

☐ 450 ☐ 550 ☐ 650 ☐ 750 ☐ OTHER

b. Is it networked with other Prime computers?

☐ YES ☐ NO

c. Is it networked with any of these?

☐ IBM ☐ CDC ☐ UNIVAC ☐ HONEYWELL

d. Which of these software packages do you use?

<input type="checkbox"/> FORTRAN	<input type="checkbox"/> COBOL	<input type="checkbox"/> BASIC/VM
<input type="checkbox"/> FORTRAN 77	<input type="checkbox"/> PL/I-G	<input type="checkbox"/> POWER
<input type="checkbox"/> MIDAS	<input type="checkbox"/> DBMS	<input type="checkbox"/> SPSS
<input type="checkbox"/> RPGII	<input type="checkbox"/> FORMS	<input type="checkbox"/> PRIMENET
<input type="checkbox"/> RJE	<input type="checkbox"/> PASCAL	<input type="checkbox"/> OAS
<input type="checkbox"/> DBG	<input type="checkbox"/> DPTX	

e. Have you read any other Prime documents?

☐ YES ☐ NO

f. If YES, which ones? \_\_\_\_\_

4. Are you presently evaluating Prime?  
Is the documentation playing a part?

☐ YES ☐ NO

☐ YES ☐ NO

5. What book are you reviewing? \_\_\_\_\_

6. My initial reaction to this book was:

☐ EXCELLENT ☐ GOOD ☐ FAIR

☐ VERY GOOD ☐ FAIR

7. After reading it my reaction was:  
If BETTER or WORSE why?

☐ BETTER ☐ THE SAME ☐ WORSE

8. How often have you used this book?

☐ EVERY DAY ☐ FAIRLY OFTEN

☐ VERY OFTEN ☐ JUST GOT IT

9. Did the book have the content you expected?

☐ YES ☐ NO

If NO, why? \_\_\_\_\_

10. Did you find the organization useful?  
If NO, why? \_\_\_\_\_

☐ YES ☐ NO



- 19. Any other comments:**

- ☐
- YES
- ☐
- NO

Blank lined paper.



20. What book don't we offer that you'd like to see?

Thank you for filling out the survey.  
Check off which Programmer's Companion you would like to receive.

- |  |  |
|--|--|
| <input type="checkbox"/> PRIMOS        | <input type="checkbox"/> FORTRAN         |
| <input type="checkbox"/> FORTRAN 77    | <input type="checkbox"/> BASIC/VM        |
| <input type="checkbox"/> ASSEMBLY      | <input type="checkbox"/> POWER           |
| <input type="checkbox"/> ADMINISTRATOR | <input type="checkbox"/> WORD PROCESSING |



First Class Permit #531 Natick, Massachusetts 01760

## BUSINESS REPLY MAIL

Postage will be paid by:

# PRIME

Attention: Technical Publications  
Bldg 10B  
Prime Park, Natick, MA 01760

NO POSTAGE  
NECESSARY  
IF MAILED  
IN THE  
UNITED STATES



